ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
INGENIERÍA DEL SOFTWARE

Aplicación multiplataforma para la consulta y gestión de contenido multimedia

Multiplatform service for consultation and manage multimedia content

Realizado por
Antonio Romero Gómez
Tutorizado por
Eduardo Guzmán De los Riscos
Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Diciembre 2016

Fecha defensa:
El Secretario del Tribunal

Resumen: En la actualidad consumimos más de 8 horas diarias de contenido multimedia, pero siempre que nos disponemos a ver una nueva película o serie, se nos plantea el mismo dilema: ¿Qué película o serie ver? ¿Será buena? etc. Es habitual perder mucho tiempo eligiendo, ya que para ello solemos buscar valoraciones, comentarios en diferentes web y aplicaciones, pero a menudo nos equivocamos. Este proyecto pretende paliar este problema, proporcionando información relevante de calidad, para reducir al mínimo el tiempo decisión y el número de equivocaciones. Para ello se ha creado una plataforma unificada, a la que se podrá acceder, a través, de las aplicaciones nativas de Android e IOS, además de la aplicación web. La plataforma ofrece al usuario un sistema de clasificación de películas intuitivo y rápido, que mantendrá ordenado el contenido, en listas de favoritos, vistos, pendientes y lista negra. Cada película tendrá asociada una puntuación media que se actualizará, en función, de las puntuaciones de los sitios más relevantes: IMDb, filmaffinity, RottenTomatoes, TVISO, Metacritic, etc. Además de ofrecer a simple vista información básica de una película: el título, la portada y su puntuación media, se podrá ver información más detallada como: sinopsis, género, puntuaciones, duración, año, participantes. Adicionalmente, se pondrán consultar las películas en la que ha trabajado cualquiera de los participantes de la película consultada. Finalmente, el usuario podrá buscar a otros y seguirlos, pudiendo consultar su lista de películas favoritas y su información básica.

Palabras claves: móvil, aplicación, script, araña, películas, multimedia, animación, IOS, Swift, Django, Python, API, REST, TVISO, Metacritic, PostgrestSQL, IMDb, filmaffinity, RottenTomatoes.

Abstract: Actually we consume more than eight hours daily in multimedia. When we are prepared to watch a new movie or TV show, frequently we ask ourselves some questions such as: ¿What movie/show can we watch? or ¿Does this movie/show worth? In order to solve these issues, we could search for movies reviews o comments in different websites and applications, what sometimes is a waste of time and we probably did not choose the right film or series. This project tries to palliate this problem, providing some relevant quality information to minimize the decision time and the number of mistakes committed. Therefore, a unified platform has been created that can be accessed through native applications developed for Android and IOS systems and, also, through the web application. Each film will have associated an average scoring that will be updated according to the score taken from the most relevant websites, i.e. IMDb, filmaffinity, Rotten Tomatoes, TVISO, Metacritic, among others. Furthermore, to offer the basic information about the film, as the title, the cover and the average scoring, at first sight, we could also see more information in detail, as the synopsis, the film genre, the scoring, the length, the first release and the participants of the film. There is also the possibility to search the movies, as well as to look up the film performed by anyone of its participants. This platform has a social nature, because any user could look for other users and follow them, and also consult their favorites films and their basic information.

Keywords: mobile, application, script, scrapers, movie, film, multimedia, animation, swipe, IOS, Swift, Django, Python, API, REST, TVISO, Metacritic, PostgrestSQL, IMDb, filmaffinity, RottenTomatoes

Capítulo 1. Introducción

1.1. Motivación

En las últimas décadas se han producido diversos hitos en la historia, que han favorecido el avance de las tecnologías. Se puede destacar alguno de ellos, como el primer ordenador personal, la posibilidad de acceder a internet desde los hogares o dispositivos móviles, el incremento de la velocidad de las conexiones en todo el mundo y la aparición de los “dispositivos inteligentes”.

La penetración de las redes 3G/4G supuso un gran avance, ya que hasta entonces, las redes móviles no tenían la suficiente velocidad para favorecer el uso de dispositivos móviles, de la forma, en que los usamos en la actualidad. Esto unido a la facilidad de hacer cualquier gestión en cualquier sitio, sin la necesidad de un ordenador, provocó un cambio de tendencias en todo el mundo.

Si observamos la Figura 1.1, se puede afirmar, que en los últimos años se ha incrementado vertiginosamente la venta de dispositivos móviles. Como consecuencia de esta demanda y la gran competencia existente, las empresas del sector se han visto obligadas a realizar una gran inversión en I+D+I, para que sus dispositivos sean los mejores y más vendidos, además de cubrir las necesidades y exigencias del mercado.

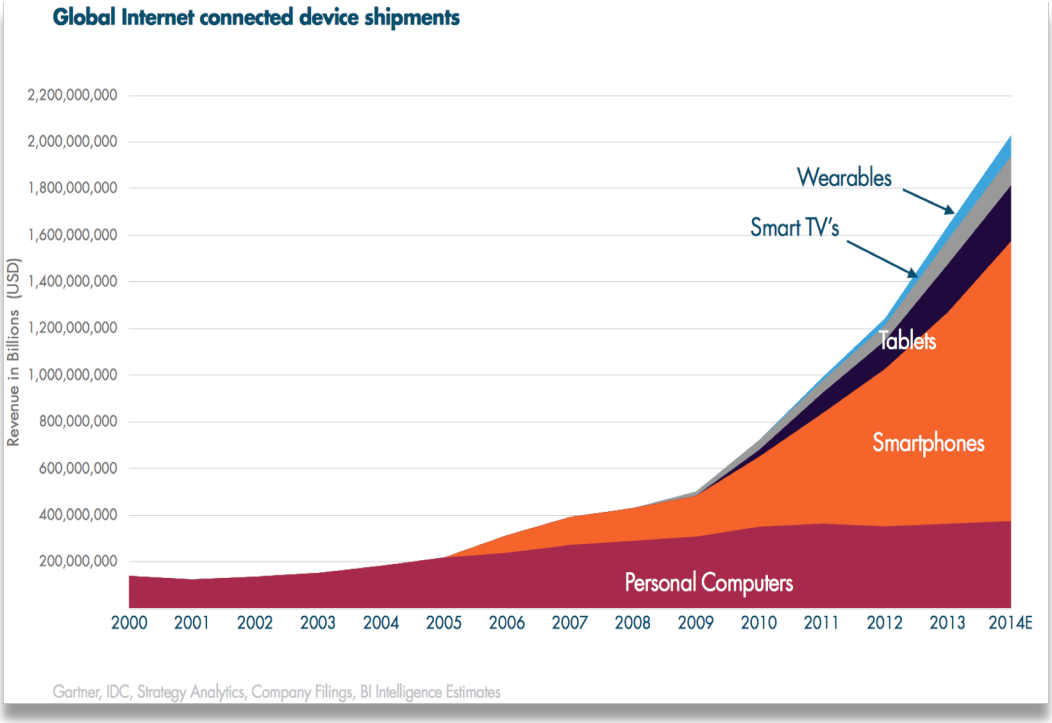

Figura 1.1 Dispositivos conectados a internet (Percolate, 2015)

Esta gran cantidad de dispositivos conectados, ha provocado un gran incremento del consumo de datos en los dispositivos móviles. Como podemos ver en la Figura 1.2, la mayor parte de los datos consumidos son vídeos. Además, según las predicciones, el consumo de vídeos va experimentar un gran crecimiento en los próximos años.

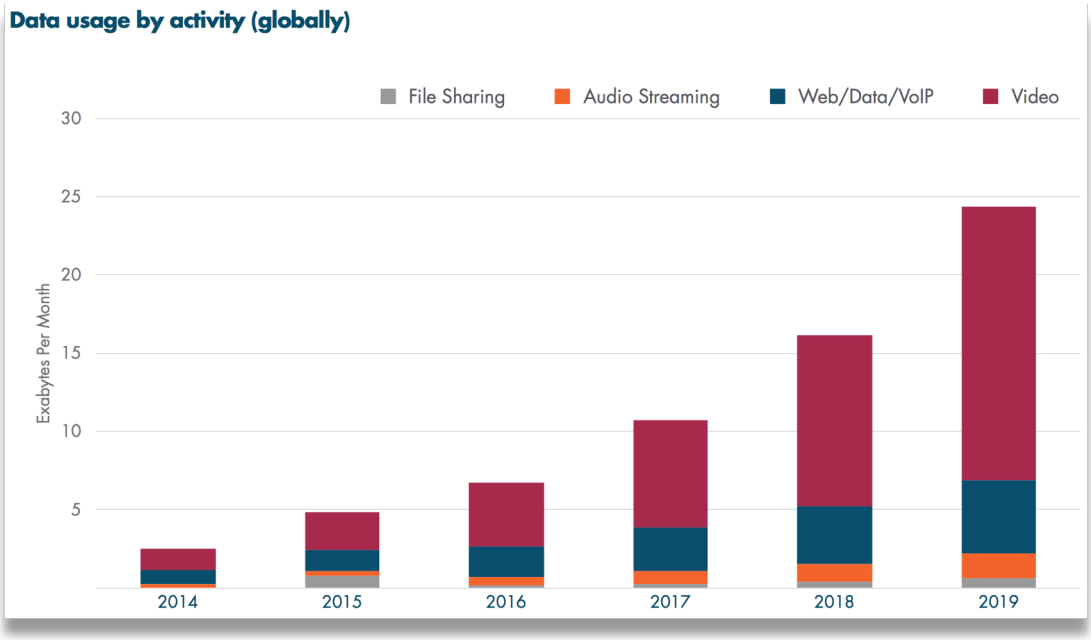

Figura 1.2 Uso de datos clasificados según actividad (Percolate, 2015)

Según un estudio (ZenithOptimedia, 2015), consumimos de media 8 horas de contenido multimedia al día. Es normal encontrar una gran cantidad de información sobre cualquier tema concreto, pero se encuentra muy fragmentada. Debido a este hecho, cuando nos disponemos a ver una película o serie, en muchas ocasiones no sabemos qué elegir.

Para seleccionar el contenido multimedia, tenemos en cuenta las puntuaciones que nos ofrecen algunas webs de carácter social, en las cuales, los usuarios incluyen comentarios y recomendaciones sobre películas y series, que en muchos casos no suelen ser muy acertadas. Además, se suele dar la situación de que el contenido de diversas plataformas se repite, por tanto, nos vemos obligados a consultar y recabar información de muchas páginas distintas, en lugar de estar localizada en un solo sitio.

En definitiva, es habitual que perdamos mucho tiempo en decidir y si añadimos que quizás seamos varias personas las que vamos a ver la película, será más complicado aún.

1.2. Objetivos

A raíz de estos problemas y la gran cantidad de información fragmentada en Internet, surgió la idea de desarrollar una plataforma de búsqueda y comparación de puntuaciones de películas y series, proporcionando información relevante de calidad. En ella, el usuario puede centralizar toda su información en películas y series, reduciendo al mínimo el tiempo decisión y el número de equivocaciones.

La plataforma ofrece al usuario, un sistema de clasificación de películas intuitiva y rápida, que mantendrá ordenada las preferencias del usuario, con el menor esfuerzo posible. Las películas se pondrán clasificar en diferentes listas:

- Lista de **favoritos**, para sus películas favoritas.
- Lista de **visto**, para las películas que haya visto, pero no lleguen a ser sus favoritas.
- Lista de **pendientes**, para las películas que quiera ver próximamente.
- Lista **negra**, para películas que no les interesen, las haya o no visto previamente

Cada película tendrá asociada una puntuación media que se actualizará, en función, de las puntuaciones de los sitios más relevantes. En el sistema de clasificación se mostrará información básica de la película: el título, la carátula y su puntuación media. Esta información se podrá ampliar, ya que se tiene acceso a una vista de detalle, que mostrará la sinopsis, género, puntuaciones, duración, año, participantes, etc.

El usuario podrá examinar las películas en las que ha trabajado cualquier miembro del reparto de la película consultada y se ofrecerá un sistema de búsqueda para películas y usuarios. Existirá la posibilidad de seguir a otros usuarios, teniendo acceso a su listado de películas favoritas e información básica.

Para cumplir estos objetivos y funcionalidades, se ha utilizado Django para el servidor, para la base de datos se ha utilizado PostgreSQL y se ha implementado una API Rest para la comunicación entre las aplicaciones y el servidor.

Para cubrir el mayor número de usuarios posible, se han desarrollado aplicaciones nativas para Android e IOS, plataformas que cubren el 99% de los “teléfonos inteligentes”. Además se ha desarrollado un sitio web quedando, de esta forma, cubierto el 100% de los usuarios potenciales. Las tres aplicaciones tendrán disponibles los idiomas inglés y español, estando preparadas para añadir nuevos idiomas fácilmente.

1.3. Materiales y tecnología usada.

- Hardware:
    - Ordenador Sobremesa, i5 6500, con Linux Ubuntu 16.04
    - Portátil MacBook Pro, con MacOS Sierra
    - Teléfono móvil IPhone 5 IOS 9-10.
    - Pantalla Benq 24”
- Software:
    - Herramienta de desarrollo Xcode 7 y 8
    - Administrador de paquetes para Mac Homebrew,
    - Administrador de paquetes para Xcode Carthage.
    - Administrador de paquetes de Python pip.
    - Administrador de paquetes para Linux apt-get.
    - Navegador Chrome, extensión Rest Client
    - Editor de texto Atom.
    - Control de versiones git y repositorio online Github
    - Herramienta de gestión de proyecto Trello
    - Herramienta de comunicación Slack, Hangout, Skype

1.4. Contenido de la memoria

La memoria del TFG ha sido dividida en varios capítulos, teniendo en cuenta los temas más importantes de cada fase de desarrollo:

- **Capítulo 1. Introducción**: En este capítulo se hace una breve introducción del proyecto, explicando las necesidades y los motivos por lo que se ha realizado, además de los objetivos del mismo.
- **Capítulo 2. Herramientas y Tecnologías empleadas**: Se enumerarán y describirán las diferentes tecnologías usadas y se realizará una breve introducción a las mismas.
- **Capítulo 3. Especificación de requisitos**: Está formado por el análisis de requisitos realizado, se divide en requisitos funcionales y no funcionales. Estos representaran todas las funciones que debe cumplir el proyecto.
- **Capítulo 4. Análisis y diseño**: Tras el análisis de requisitos, en este capítulo, se extraen los casos de uso correspondientes y se define la arquitectura del sistema y la estructura del servidor.
- **Capítulo 5. Implementación e Instalación**: Será el punto más extenso de la memoria. En él se explicará tanto el desarrollo de la aplicación IOS, como la del servidor. El servidor constará de varios puntos clave: la creación del mismo, la base de datos, los scripts, que recopilan los datos de las películas y la API REST que da servicio a las aplicaciones móviles. Se mostrarán las vistas de la aplicación y se explicará su funcionamiento.
- **Capítulo 6. Conclusiones y trabajo futuro**: En este último capítulo, expondremos los resultados y argumentaremos las conclusiones extraídas de todo el proceso. Finalmente se comentará posibles ampliaciones y mejoras del proyecto.
- **Bibliografía**: En este apartado, se recopilará toda la bibliografía que se ha utilizado para la realización de este proyecto, incluida la elaboración de la memoria.

1.5. División del trabajo

El proyecto se ha realizado en grupo con otros dos compañeros. Consta de una parte común, el servidor Django, y una parte individual, en mi caso la aplicación IOS. En la Tabla 1.1 podemos ver el porcentaje realizado por cada miembro en cada una de las partes del entorno que se ha desarrollado.

			Jesús Garrido Moscoso	José Antonio Palacios Ramírez	Antonio Romero Gómez
Servidor Django	Creación e instalación		10%	80%	10%
	API	Movie	30%	40%	30%
		User	10%	80%	10%
	Scripts	Scrappers	45%	10%	45%
		Tviso	45%	10%	45%
		Trakt.tv	45%	10%	45%
		Script principal	40%	20%	40%
Aplicación Android			100%		
Aplicación IOS					100%
Aplicación Web				100%	

Tabla 1.1 División del Trabajo

Capítulo 2. Herramientas y tecnologías empleadas

2.1. Python

Es un lenguaje de programación interpretado se caracteriza por tener una sintaxis peculiar: no utiliza llaves para delimitar funciones y bloques de código, para este fin utiliza las los “dos puntos” y las tabulaciones, el objetivo de esto es favorecer la legibilidad del código.

Soporta programación orientada a objetos, programación imperativa y algo de programación funcional. Su tipado es dinámico, no hace falta declarar las variables previamente, y estas se pueden utilizar para tipos diferentes.

Posee una licencia PSFL, que a partir de la versión 2.1.1, es compatible con la licencia GNU. Para el proyecto hemos utilizado Python 3 que es la última versión estable del lenguaje.

Para la instalación y gestión de paquetes software escritos en Python, se utiliza el gestor de paquetes “pip”. Es muy fácil de usar, simplemente escribimos en el terminal, “pip install” el nombre del paquete.

2.2. Django

Es un framework gratuito y open source, escrito en Python. Posee un conjunto de componentes, que facilitan el desarrollo de nuevas aplicaciones.

Al crear una nueva aplicación, ya sea web o nativa, hay ciertas funcionalidades en un servidor que vamos a necesitar repetidamente. Django facilita esta parte del desarrollo aportando una serie de componentes predefinidos, entre ellos, nos proporciona:

- Un sistema para gestionar la autenticación de los usuarios, registro, inicio de sesión, etc.
- Un panel de administración para gestionar tu aplicación.
- Conexión con la base de datos y la posibilidad de usar su ORM, para operar con la base de datos.
- Componente para ayudar a gestionar la subida de archivos.

Además, existen multitud de componentes externos que pueden ayudarnos con otras partes del desarrollo e incluso podemos crear nuestros propios componentes. Esto facilita la reutilización de código y permite centrarnos en las funcionalidades específicas de la aplicación que estemos desarrollando.

Actualmente estamos utilizando Django 1.8, que es la última versión estable. Para ejecutar el servidor es tan simple como, “python3 manage.py runserver”. De esta forma, solo se ejecutará en local, por defecto en el puerto 8000; si añadimos 0.0.0.0: Puerto, se ejecutará para estar disponible en toda la red. Se ha elaborado un documento con las instrucciones para instalar todo lo necesario, configurarlo y poner en funcionamiento este servidor.

2.3. API REST

REST, Representational State Transfer, es un tipo de arquitectura que utiliza el protocolo estándar HTTP. Se creó en el año 2.000 por Roy Fielding, coautor también de la especificación HTTP.

Nos permite crear servicios para cualquier aplicación, con independencia de qué tipo de dispositivo, el lenguaje que se haya utilizado o el sistema operativo donde esté funcionando; solo es necesario que soporte la especificación HTTP.

2.4. Django Rest Framework

Es un componente externo, para Django, que nos “facilita” la creación de la API REST que utilizaremos para la comunicación entre las aplicaciones y el servidor.

Actualmente utilizamos la versión 3; las clases más importantes de este framework son:

- Los ViewSets, que son los controladores, donde se define la lógica de cada servicio de la API.
- Los Serializers, que tienen una doble función, serializar los datos que provienen de las llamadas a la API y para las respuestas. Se utiliza para convertir los datos serializados en datos soportados por la especificación http, en nuestro caso JSON.

2.5. Base de datos PostgreSQL

Es un sistema de gestión de bases de datos relacional, que es una extensión de los sistemas relacionales tradicionales. Añade características de la programación orientada a objetos. Se distribuye bajo la licencia BSD y su código está disponible libremente. Es el sistema de código abierto más potente y en sus últimas versiones puede llegar a competir con sistemas comerciales. Utiliza un sistema multiproceso, en vez de multihilo, característica que mejora su estabilidad; si falla algún proceso, no afectará al resto.

2.6. Herramienta de desarrollo Xcode.

Es el entorno de desarrollo integrado de Apple. Se ofrece gratuitamente con el sistema MacOS. Se pueden desarrollar aplicaciones en varios lenguajes; incluye una colección de compiladores del proyecto GNU y puede compilar código en C, C++, Swift, Objective-C, Objective-C++, Java y AppleScript.

Consta de un sistema de depuración muy completo en el que, además de las opciones habituales de otros IDEs, podemos monitorizar los recursos consumidos por la aplicación, memoria RAM, ROM, actividad de red, etc. Permite optimizar las aplicaciones para que consuman menos recursos.

2.7. Gestor de paquetes Carthage

Es un gestor de dependencias para Xcode; hay otras opciones como CocoaPods, pero esta es una opción más flexible y menos intrusiva. El uso es parecido a la de la mayoría gestores de paquetes. En el directorio principal del proyecto, tendremos un archivo denominado Cartfile, en él se recogerán todas las dependencias. Podremos instalar y actualizar las dependencias con el comando “carthage update”. Esto descargará las dependencias para todas las plataformas de Apple. Si especificamos la plataforma añadiendo al final “–platform IOS”, solo se descargarán para esa plataforma.

Las principales librerías externas utilizadas en la aplicación IOS son: Alamofire, para las llamadas a los servicios de la API REST, Kingfisher para la descarga y el almacenamiento en caché de imágenes.

2.8. Swift

La aplicación de IOS se ha desarrollado en el lenguaje Swift, lenguaje por el que apuesta Apple para todos sus dispositivos, dejando atrás ya, al veterano Objective-c.

Este lenguaje soporta programación orientada a objetos y, a partir de Swift 2, se introdujo un nuevo paradigma, la programación orientada a protocolos, que intenta complementar las carencias y problemas que tienen la programación orientadas a objetos. Las piezas claves de esta nueva forma de programar son:

- **Los protocolos**: son plantillas de especificación que definimos para crear unas reglas determinadas, parecido a las interfaces de la orientación a objetos, pero estos se pueden aplicar varias clases, estructuras e incluso enumeraciones.
- **Las extensiones**: aplicada a los protocolos, nos permiten generar la implementación por defecto de los métodos o propiedades especificadas en el protocolo.

Desde que Apple en 2015 liberase el código, Swift es un lenguaje de código abierto. Este cambio de rumbo, ha abierto un nuevo abanico de posibilidades. Pretende llegar a ser un lenguaje multiplataforma, ya es posible programar en Swift bajo Linux. Empresas tan relevantes como IBM están empezando a utilizar este lenguaje en sus servidores.

2.9. Git, GitHub y Git-Flow

Sistema de control de versiones, diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones de un gran número de archivos de código fuente.

GitHub es una plataforma de desarrollo colaborativo de software para alojar proyectos usando el sistema de control de versiones Git. Hemos utilizado un repositorio privado, que hemos obtenido gratuitamente por ser estudiantes.

Git-Flow, es una guía de buenas prácticas para organizar las ramas de un repositorio. El esquema básico consta de dos ramas principales: master y develop, y 3 temporales: Feature, Hotfix y Realease, conocidas como ramas de soporte. Existe un comando, “git-flow”, que facilita la creación del esquema básico de esta estructura de ramas, además de poder gestionar la creación y gestión de ramas siguiendo esta estructura. En la Figura 2.1 podemos observar la estructura.

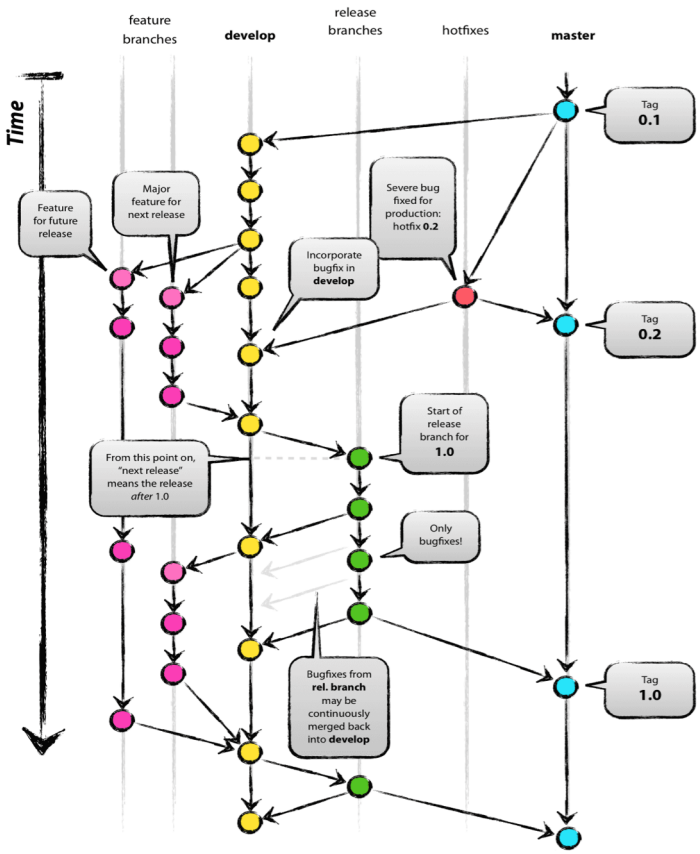

Figura 2.1 Estructura Git-Flow (Driessen, 2010)

Capítulo 3. Especificación de requisitos

Es el conjunto de técnicas y procedimientos que nos permiten definir los servicios que debe ofrecer el sistema y sus restricciones. Esta descripción del sistema, entre otras características, tiene que ser completa, consistente y debe evitar ambigüedades. Los requisitos suelen clasificarse en dos grupos:

- **Requisitos funcionales**: La mayoría de los requisitos suelen ser de este tipo, ya que son los definen todas los servicios y respuestas que debe cumplir el sistema.
- **Requisitos no funcionales**: En ellos se definen todos los requisitos que no se refieren directamente a las funciones del sistema, sino a sus propiedades, como la fiabilidad, la velocidad de respuesta, la capacidad, la seguridad, etc.

3.1. Recopilación de datos

Nuestro proyecto es particular, ya que no hay un cliente que nos demande la creación de este proyecto, sino es más una idea, que surge por iniciativa propia con el objetivo de proporcionar un entorno de herramientas que ayuden al usuario con la selección de una nueva película o serie para ver.

Hemos comenzado investigando qué aplicaciones relacionadas con el cine existían en ese momento, siendo usuarios durante un periodo de tiempo, para descubrir qué posibles funcionalidades podríamos incluir, con el objetivo de que nuestro proyecto se diferenciara de lo que existía actualmente en el mercado.

Una de las grandes preocupaciones era de dónde íbamos a recopilar todos los datos sobre las películas, para tener una base de datos completa y, con ella, poder ofrecer un buen servicio. Nos hemos puesto en contacto, intercambiando varios correos con los sitios más relevantes de puntuaciones tales como IMDb, filmaffinity, Tviso, Sensacine, Metacritic, RottenTomatoes, etc. En los contactos que hemos realizado, hemos encontrado diversas situaciones: Algunos tenían API y nos han facilitado el acceso; otros no tenían, pero nos han dado permiso para hacer un scrapper que recopile los datos de su web. Por último, algunos sitios no nos han contestado e incluso, en el caso de RottenTomatoes, nos ha llegado a pedir más de 80.000 $ por utilizar su API.

3.2. Aplicación IOS

3.2.1. Requisitos funcionales

RF01 - Registro de usuario: El usuario debe poder registrarse en la aplicación.

- **RF01.1 -** El usuario podrá registrarse en la aplicación rellenando los campos obligatorios de usuario, correo electrónico y contraseña.
- **RF01.2 -** El sistema mostrará dos campos de contraseña en el registro, para evitar equivocaciones.
- **RF01.3 -** El sistema mostrará un mensaje de error cuando los dos campos de contraseñas no coincidan.
- **RF01.4 -** El sistema mostrará un mensaje de error si el nombre de usuario elegido está ya siendo utilizado por otro usuario registrado.
- **RF01.5 -** El sistema mostrará un mensaje de error, si el correo electrónico indicado está ya siendo usado por otro usuario.
- **RF01.6 -** El sistema creará el perfil con el idioma configurado en el móvil.
- **RF01.7 -** El sistema creará el perfil con un avatar por defecto.
- **RF01.8 -** El sistema inicializara por defecto los campos opcionales.

RF02 - Inicio de sesión: El usuario debe poder iniciar su sesión en la aplicación.

- **RF02.1 -** El usuario debe poder iniciar su sesión, introduciendo su usuario y contraseña, con los que previamente se haya registrado.
- **RF02.2 -** El usuario debe poder iniciar su sesión, introduciendo su correo electrónico y contraseña, con los que previamente se haya registrado.
- **RF02.3 -** El sistema mostrará un mensaje de error, cuando el usuario/email o la contraseña sean incorrectas.

RF03 - Cierre de sesión: El usuario debe poder cerrar la sesión.

- **RF03.1 -** El sistema borrará todos los datos del usuario guardados en la aplicación.

FR04 - Sistema de clasificación de películas: El usuario podrá clasificar las películas, según su preferencia.

- **FR04.1 - Sistema de clasificación swipe**: El usuario debe poder clasificar las películas mostradas, de una manera rápida e intuitiva
    1. **FR04.1.1 -** El usuario podrá clasificar la película mostrada como favorita, deslizando su carátula hacia arriba.
    2. **FR04.1.2 -** El usuario podrá clasificar la película mostrada como pendiente, deslizando su carátula a la izquierda.
    3. **FR04.1.3 -** El usuario podrá clasificar la película mostrada como vista, deslizando su carátula a la derecha.
    4. **FR04.1.4 -** El usuario podrá clasificar la película mostrada como como no interesante, deslizando su carátula hacia abajo.
- **FR04.2 - Sistema de clasificación mediante botones:** El usuario podrá clasificar sus películas pulsando el botón correspondiente.

1. **FR04.2.1 -** El usuario podrá clasificar la película mostrada pulsando el botón de favoritos.
2. **FR04.2.2 -** El usuario podrá clasificar la película mostrada pulsando el botón de pendientes.
3. **FR04.2.3 -** El usuario podrá clasificar la película mostrada pulsando el botón de vistas.
4. **FR04.2.4 -** El usuario podrá clasificar la película mostrada pulsando el botón de no interesante.

RF05 - Barra de pestañas: El usuario podrá acceder a todas las secciones de la aplicación fácilmente a través de la barra de pestañas.

RF06 - Consular perfil: El usuario podrá consultar los datos de su perfil.

- **RF06.1 -** El usuario podrá ver su nombre de usuario.
- **RF06.2 -** El usuario podrá ver su correo electrónico.
- **RF06.3 -** El usuario podrá ver su género.
- **RF06.4 -** El usuario podrá ver su fecha de nacimiento.
- **RF06.5 -** El usuario podrá ver su país.
- **RF06.6 -** El usuario podrá ver sus seguidores.
- **RF06.7 -** El usuario podrá ver los usuarios que sigue.
- **RF06.8 -** El usuario podrá ver su avatar.

RF07 - Editar perfil: El usuario podrá editar cualquiera de los datos de su perfil.

- **RF07.1 -** El usuario podrá modificar su nombre de usuario.
- **RF07.2 -** El usuario podrá modificar su correo electrónico.
- **RF07.3 -** El usuario podrá modificar su género.
- **RF07.4 -** El usuario podrá modificar su fecha de nacimiento.
- **RF07.5 -** El usuario podrá modificar su país.
- **RF07.6 -** El usuario podrá modificar el idioma de su perfil.
- **RF07.7 -** El usuario podrá modificar su avatar.
- **RF07.8 -** El sistema mostrará un mensaje de error si el nombre de usuario elegido, está ya utilizado por otro usuario registrado.
- **RF07.9 -** El sistema mostrará un mensaje de error, si el correo electrónico indicado, está ya usado por otro usuario.

RF08 - Visualizar las listas de películas clasificadas.

- **RF08.1 -** El usuario podrá visualizar una vista general, donde se mostrará las últimas películas clasificadas de cada lista.
- **RF08.2 -** El usuario podrá visualizar todas las películas clasificadas en las listas de:
    1. Favoritas.
    2. Vistas
    3. Pendientes
    4. No deseadas (lista negra)
- **RF08.3 -** El sistema mostrará las últimas películas clasificadas primero.

- **RF08.4 -** El sistema representará cada película, con su carátula, título y una puntuación media.

RF09 - Visualizar la información detallada de una película.

- **RF09.1 -** El usuario podrá ver un listado con todas las puntuaciones de los sitios más relevantes.
- **RF09.2 -** El usuario podrá ver el título de la película.
- **RF09.3 -** El usuario podrá ver la duración de la película.
- **RF09.4 -** El usuario podrá ver el año de la película.
- **RF09.5 -** El usuario podrá ver los géneros de la película.
- **RF09.6 -** El usuario podrá ver la sinopsis de la película.
- **RF09.7 -** El usuario podrá ver los productores de la película.
- **RF09.8 -** El usuario podrá ver un listado con todos los participantes, directores, actores, escritores, etc.
- **RF09.9 -** El usuario podrá consultar las películas, donde han trabajado sus participantes.
- **RF09.9 -** El sistema mantendrá actualizadas las puntuaciones de los diferentes sitios.
- **RF09.10 -** El sistema recalculará la media de las puntuaciones cuando estas sean actualizadas.

RF10 - Búsqueda de películas: El usuario podrá buscar las películas por título.

- **RF10.1 -** El sistema realizará la búsqueda por el título original o en el idioma del perfil de usuario.

RF11 - Soporte multilenguaje: El usuario tendrá disponible la aplicación tanto en inglés como en español.

- **RF11.1 -** El sistema mostrará la interfaz de usuario en el idioma que tenga configurado el teléfono. Si es diferente a los lenguajes disponibles, por defecto se mostrará en inglés.
- **RF11.2 -** El sistema mostrará los datos de las películas en el idioma que se haya configurado en el perfil.

RF12 - El sistema mostrará un mensaje de error cuando la aplicación no pueda conectar con el servidor.

3.2.2. Requisitos no funcionales

RNF01 - Las claves se guardarán cifradas en la base de datos.

RNF02 - Se utilizará un sistema de autenticación mediante la utilización de token, que se generará en función del usuario y la contraseña.

RNF03 - Las comunicaciones entre el servidor y las aplicaciones se harán utilizando el protocolo de aplicación HTTPS, para asegurar un canal seguro para las comunicaciones de datos.

Capítulo 4. Análisis y diseño

4.1. Casos de uso, aplicación IOS.

En esta sección mostraremos los casos de uso que se han generado, a partir del análisis de requisitos que se ha expuesto en el capítulo anterior.

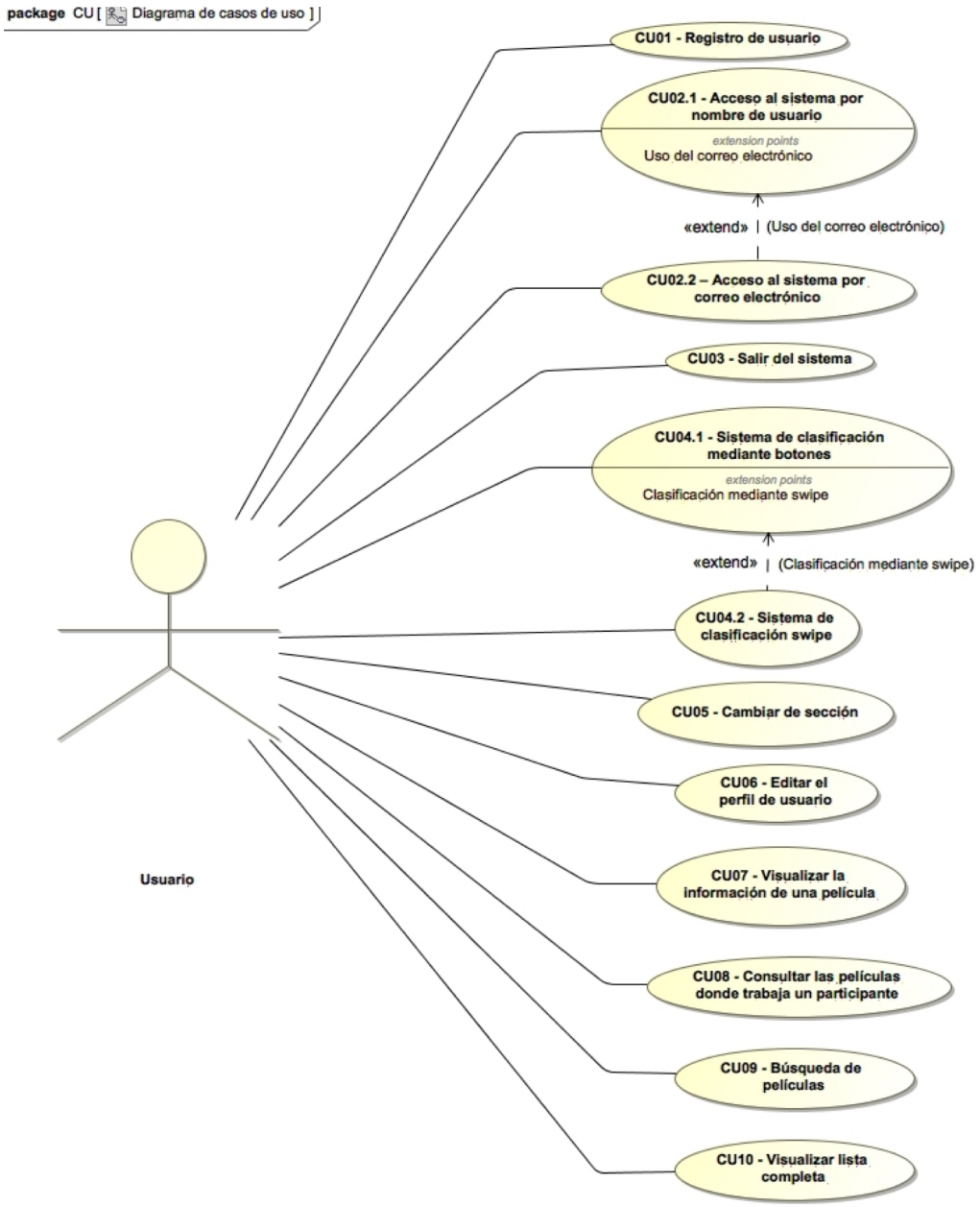

Figura 4.1 Diagrama de casos de uso de la aplicación IOS

CU01 – Registro de usuario:

- **Descripción:** El usuario podrá registrarse en el sistema, rellenando los campos de usuario, correo electrónico y contraseña.
- **Actores:** Usuario
- **Pre-condición:** Ninguna
- **Post-condición:** El usuario se registra en el sistema correctamente
- **Prioridad:** Alta
- **Escenario principal:**
    1. El usuario pulsa el botón para crear una cuenta.
    2. El sistema muestra el formulario de registro.
    3. El usuario introduce el nombre.
    4. El sistema comprueba que el tamaño del usuario es de más de 5 caracteres.
    5. El usuario introduce su correo electrónico.
    6. El sistema comprueba que el correo tiene un formato correcto.
    7. El usuario introduce una contraseña.
    8. El sistema comprueba que el tamaño de la contraseña es de más de 5 caracteres.
    9. El usuario introduce la confirmación de la contraseña.
    10. El sistema comprueba que las dos contraseñas coinciden
    11. El usuario pulsa el botón para registrarse.
    12. El sistema comprueba que todos los campos son válidos.
    13. El sistema creará la cuenta e iniciará la sesión automáticamente.
- **Escenario alternativo:**

    4.a) El sistema comprueba que el tamaño del usuario es menor a 5 caracteres.

    4.a.1) El sistema muestra un mensaje en rojo, encima del campo de usuario, indicando que el usuario debe tener al menos 5 caracteres.

    6.a) El sistema comprueba que el formato del correo es incorrecto.

    6.a.1) El sistema muestra un mensaje en rojo, encima del campo de correo, indicando que el correo tiene un formato incorrecto.

    8.a) El sistema comprueba que el tamaño de la contraseña es menor de 5 caracteres.

    8.a.1) El sistema muestra un mensaje en rojo, encima del campo de contraseña, indicando que la contraseña debe ser al menos de 5 caracteres

    10.a) El sistema comprueba que las contraseñas son diferentes.

    10.a.1) El sistema muestra un mensaje en rojo, encima del campo de confirmar contraseña, indicando que las contraseñas son diferentes.

    12.a.1) El sistema comprueba que hay campos que son inválidos.

    12.a.1.1) El sistema comprueba que el nombre de usuario, ya está utilizado por otro usuario registrado.

12.a.1.1.1) El sistema muestra un mensaje de error, indicando que el nombre de usuario ya está utilizado.

12.a.1.2) El sistema comprueba que el correo ya está usado por otro usuario.

12.a.1.2.1) El sistema muestra un mensaje de error, indicando que el correo electrónico ya está utilizado.

12.a.2) El sistema no tiene conexión con el servidor.

12.a.2.1) El sistema muestra un mensaje de error, indicando que se ha perdido la conexión con el servidor.

CU02.1 - Acceso al sistema por nombre de usuario:

- **Descripción:** El usuario podrá acceder al sistema introduciendo, su nombre de usuario y contraseña.
- **Actores:** Usuario
- **Pre-condición:** El usuario tendrá que estar registrado en el sistema
- **Post-condición:** El usuario accederá al sistema
- **Prioridad:** Alta
- **Escenario principal:**
  1. El usuario introduce su usuario.
  2. El usuario introduce su contraseña.
  3. El usuario pulsa el botón de inicio de sesión.
  4. El sistema comprueba que el usuario y la contraseña son correctos.
  5. El sistema muestra la vista de clasificación swipe.
- **Escenario alternativo:**

  4.a.1) El sistema comprueba que el usuario o la contraseña son incorrectos

  4.a.1.1) El sistema muestra un mensaje de error, indicando que el usuario o la contraseña son incorrectos.

  4.a.2) El sistema no tiene conexión con el servidor.

  4.a.2.1) El sistema muestra un mensaje de error, indicando que se ha perdido la conexión con el servidor.

CU02.2 – Acceso al sistema por correo electrónico:

- **Descripción:** El usuario podrá acceder al sistema introduciendo, su correo electrónico y contraseña.
- **Actores:** Usuario
- **Pre-condición:** El usuario tendrá que estar registrado en el sistema
- **Post-condición:** El usuario accederá al sistema
- **Prioridad:** Media
- **Escenario principal:**
  1. El usuario introduce su correo electrónico.
  2. El usuario introduce su contraseña.
  3. El usuario pulsa el botón de inicio de sesión.
  4. El sistema comprueba que el correo electrónico y la contraseña son correctos.
  5. El sistema muestra la vista de clasificación swipe.

- **Escenario alternativo:**

  4.a.1) El sistema comprueba si el correo electrónico o la contraseña son incorrectos

  4.a.1.1) El sistema muestra un mensaje de error, indicando que el usuario o la contraseña son incorrectos.

  4.a.2) El sistema no tiene conexión con el servidor.

  4.a.2.1) El sistema muestra un mensaje de error, indicando que se ha perdido la conexión con el servidor.

CU03 - Salir del sistema:

- **Descripción:** El usuario podrá cerrar su sesión, borrando los datos guardados en la aplicación.
- **Pre-condición:** El usuario tendrá que estar identificado en el sistema y estar en la vista de su perfil.
- **Post-condición:** El usuario saldrá del sistema y el sistema borrará los datos guardados en la aplicación.
- **Prioridad:** Alta
- **Escenario principal:**

  1. El usuario pulsa el botón de cerrar sesión.
  2. El sistema muestra un mensaje, indicando que se cerrará la sesión y se borrará todos los datos de la aplicación.
  3. El usuario pulsa el botón de aceptar.
  4. El sistema borra los datos y vuelve a la vista de inicio de sesión.

- **Escenario alternativo:**

  3.a)El usuario pulsa cancelar.

  3.a.1) El sistema cierra el mensaje.

CU04.1 - Sistema de clasificación mediante botones:

- **CU04.1.1 - Clasificar película como favorita:**
  - **Descripción:** El usuario podrá clasificar la película mostrada como favorita, pulsando el botón, favorito.
  - **Pre-condición:** El usuario tendrá que estar identificado en el sistema y estar en la vista de clasificación swipe o en la vista de detalle.
  - **Post-condición:** El usuario clasifica la película como favorita.
  - **Prioridad:** Alta
  - **Escenario principal:**

    1. El usuario pulsa el botón favorito.
    2. El sistema desplaza la carátula hacia la derecha*.
    3. El sistema clasifica la película como favorita.
    4. El sistema borra de la vista la portada y muestra la siguiente película sin clasificar*.

  - **Escenario alternativo:**

    3.a) El sistema no tiene conexión con el servidor.

* Solo en la vista de clasificación swipe

3.a.1) El sistema muestra un mensaje de error, indicando que se ha perdido la conexión con el servidor.

- **CU04.1.2 - Clasificar película como pendiente:**
  - **Descripción:** El usuario podrá clasificar la película mostrada como favorita, pulsando el botón correspondiente.
  - **Pre-condición:** El usuario tendrá que estar identificado en el sistema y estar en la vista de clasificación swipe o en la de detalle.
  - **Post-condición:** El usuario clasifica la película como pendiente.
  - **Prioridad:** Alta
  - **Escenario principal:**
    1. El usuario pulsa el botón pendiente.
    2. El sistema desplaza la portada hacia arriba*.
    3. El sistema clasifica la película como pendiente.
    4. El sistema borra de la vista la portada y muestra la siguiente película sin clasificar*.
  - **Escenario alternativo:**
    3.a) El sistema no tiene conexión con el servidor.
    3.a.1) El sistema muestra un mensaje de error, indicando que se ha perdido la conexión con el servidor.
- **CU04.1.3 - Clasificar película como vista:**
  - **Descripción:** El usuario podrá clasificar la película mostrada como vista, pulsando el botón vista.
  - **Pre-condición:** El usuario tendrá que estar identificado en el sistema y estar en la vista de clasificación swipe o en la de detalle.
  - **Post-condición:** El usuario clasifica la película como vista.
  - **Prioridad:** Alta
  - **Escenario principal:**
    1. El usuario pulsa el botón, vista.
    2. El sistema desplaza la portada hacia abajo*.
    3. El sistema clasifica la película como vista.
    4. El sistema borra de la vista la carátula y muestra la siguiente película sin clasificar*.
  - **Escenario alternativo:**
    3.a) El sistema no tiene conexión con el servidor.
    3.a.1) El sistema muestra un mensaje de error, indicando que se ha perdido la conexión con el servidor.
- **CU04.1.4 - Clasificar película como no interesante:**
  - **Descripción:** El usuario podrá clasificar la película mostrada como no interesante, pulsando el botón lista negra.
  - **Pre-condición:** El usuario tendrá que estar identificado en el sistema y estar en la vista de clasificación swipe o en la de detalle.

* Solo en la vista de clasificación swipe

- **Post-condición:** El usuario clasifica la película como no interesante.
- **Prioridad:** Alta
- **Escenario principal:**
  1. El usuario pulsa el botón, lista negra.
  2. El sistema desplaza la carátula hacia la izquierda*.
  3. El sistema clasifica la película como no interesante.
  4. El sistema borra de la vista la carátula y muestra la siguiente película sin clasificar*.
- **Escenario alternativo:**
  3.a) El sistema no tiene conexión con el servidor.
  3.a.1) El sistema muestra un mensaje de error, indicando que se ha perdido la conexión con el servidor.

CU04.2 - Sistema de clasificación swipe:

- **CU04.2.1 - Clasificar película como favorita:**
  - **Descripción:** El usuario podrá clasificar la película mostrada como favorita, deslizando su carátula hacia arriba.
  - **Pre-condición:** El usuario tendrá que estar identificado en el sistema y estar en la vista de clasificación swipe.
  - **Post-condición:** El usuario clasifica la película como favorita.
  - **Prioridad:** Media-alta
  - **Escenario principal:**
    1. El usuario desliza la carátula de la película hacia arriba, soltando al final.
    2. El sistema comprueba que se ha desplazado suficiente.
    3. El sistema clasifica la película como favorita, borra de la vista la carátula y muestra la siguiente película sin clasificar.
  - **Escenario alternativo:**
    2.a) El sistema comprueba que se ha hecho un desplazamiento insuficiente de la portada.
    2.a.1) El sistema devuelve la carátula a su posición original.
    3.a) El sistema no tiene conexión con el servidor.
    3.a.1) El sistema muestra un mensaje de error, indicando que se ha perdido la conexión con el servidor.
- **CU04.2.2 - Clasificar película como pendiente:**
  - **Descripción:** El usuario podrá clasificar la película mostrada como pendiente, deslizando su carátula hacia la izquierda.
  - **Pre-condición:** El usuario tendrá que estar identificado en el sistema y estar en la vista de clasificación swipe.
  - **Post-condición:** El usuario clasifica la película como pendiente.
  - **Prioridad:** Media-alta

* Solo en la vista de clasificación swipe

- **Escenario principal:**
  1. El usuario desliza la carátula de la película hacia la izquierda, soltando al final.
  2. El sistema comprueba que se ha desplazado suficiente.
  3. El sistema clasifica la película como pendiente, borra de la vista la carátula y muestra la siguiente película sin clasificar.
- **Escenario alternativo:**
  2.a) El sistema comprueba que se ha hecho un desplazamiento insuficiente de la carátula.
    2.a.1) El sistema devuelve la carátula a su posición original.
  3.a) El sistema no tiene conexión con el servidor.
    3.a.1) El sistema muestra un mensaje de error, indicando que se ha perdido la conexión con el servidor.

- **CU04.2.3 - Clasificar película como vista:**
  - **Descripción:** El usuario podrá clasificar la película mostrada como vista, deslizando su portada hacia la derecha.
  - **Pre-condición:** El usuario tendrá que estar identificado en el sistema y estar en la vista de clasificación swipe.
  - **Post-condición:** El usuario clasifica la película como vista.
  - **Prioridad:** Media-alta
  - **Escenario principal:**
    1. El usuario desliza la portada de la película la derecha, soltando al final.
    2. El sistema comprueba que se ha desplazado suficiente.
    3. El sistema clasifica la película como vista, borra de la vista la carátula y muestra la siguiente película sin clasificar.
  - **Escenario alternativo:**
    2.a) El sistema comprueba que se ha hecho un desplazamiento insuficiente de la carátula.
      2.a.1) El sistema devuelve la carátula a su posición original.
    3.a) El sistema no tiene conexión con el servidor.
      3.a.1) El sistema muestra un mensaje de error, indicando que se ha perdido la conexión con el servidor.

- **CU04.2.4 - Clasificar película como no interesante:**
  - **Descripción:** El usuario podrá clasificar la película mostrada como no interesante, deslizando su carátula hacia abajo.
  - **Pre-condición:** El usuario tendrá que estar identificado en el sistema y estar en la vista de clasificación swipe.
  - **Post-condición:** El usuario clasifica la película como no interesante.
  - **Prioridad:** Media-alta
  - **Escenario principal:**
    1. El usuario desliza la carátula de la película hacia la derecha, soltando al final.

2. El sistema comprueba que se ha desplazado suficiente.
3. El sistema clasifica la película como no interesante, borra de la vista la carátula y muestra la siguiente película sin clasificar.

- **Escenario alternativo:**
  2.a) El sistema comprueba que se ha hecho un desplazamiento insuficiente de la carátula.
    2.a.1) El sistema devuelve la portada a su posición original.
  3.a) El sistema no tiene conexión con el servidor.
    3.a.1) El sistema muestra un mensaje de error, indicando que se ha perdido la conexión con el servidor.

CU05 - Cambiar de sección:

- **CU05.1 - Ir a la vista, clasificación** swipe**:**
  - **Descripción:** El usuario podrá ir a la vista de clasificación swipe.
  - **Pre-condición:** El usuario tendrá que estar identificado en el sistema.
  - **Post-condición:** El usuario se encuentra en la vista de clasificación swipe.
  - **Prioridad:** Alta
  - **Escenario principal:**
    1. El usuario pulsa el botón swipe de la barra de pestañas.
    2. El sistema muestra la vista de clasificación swipe, con un listado de películas sin clasificar.
  - **Escenario alternativo:** Ninguno
- **CU05.2 - Ir a la vista, listas de películas:**
  - **Descripción:** El usuario podrá ir a la vista de lista de películas.
  - **Pre-condición:** El usuario tendrá que estar identificado en el sistema.
  - **Post-condición:** El usuario se encuentra en la vista de lista de película.
  - **Prioridad:** Alta
  - **Escenario principal:**
    1. El usuario pulsa el botón listas de la barra de pestañas.
    2. El sistema muestra la vista de lista de películas.
    3. El sistema mostrará las listas de favoritas, pendientes, vistas y lista negra, con las últimas películas clasificadas.
  - **Escenario alternativo:** Ninguno
- **CU05.3 - Ir a la vista de perfil:**
  - **Descripción:** El usuario podrá ir a la vista de perfil.
  - **Pre-condición:** El usuario tiene que estar identificado en el sistema.
  - **Post-condición:** El usuario se encuentra en la vista de perfil.
  - **Prioridad:** Alta
  - **Escenario principal:**
    1. El usuario pulsa el botón perfil de la barra de pestañas.
    2. El sistema muestra la vista de perfil.

3. El sistema cargará el nombre de usuario, el correo electrónico, el género, la fecha de nacimiento, el país, el listado con sus seguidores y un listado con los usuarios que sigue.

- **Escenario alternativo:** Ninguno

CU06 - Editar el perfil de usuario:

- **Descripción:** El usuario podrá modificar los datos de su perfil.
- **Pre-condición:** El usuario tiene que estar identificado en el sistema y estar en la vista del perfil.
- **Post-condición:** El usuario modificará su perfil correctamente
- **Prioridad:** media
- **Escenario principal:**
  1. El usuario pulsa el botón editar.
  2. El sistema muestra la vista de editar el perfil.
  3. El usuario modifica su nombre de usuario.
  4. El sistema comprueba que el nombre de usuario tiene más de 5 caracteres.
  5. El usuario modifica su correo electrónico.
  6. El sistema comprueba que el correo electrónico tiene un formato válido.
  7. El usuario modifica su nombre.
  8. El usuario modifica sus apellidos.
  9. El usuario modifica su fecha de nacimiento.
  10. El sistema comprueba que la fecha de nacimiento tiene un formato correcto.
  11. El usuario modifica su sexo.
  12. El sistema comprueba que el sexo introducido es válido.
  13. El usuario modifica su país.
  14. El sistema comprueba que el país es válido.
  15. El usuario modifica el idioma de su perfil.
  16. El usuario pulsa el botón de modificar avatar.
  17. El sistema muestra la biblioteca del móvil.
  18. El usuario elige la nueva imagen de perfil.
  19. El sistema cierra la biblioteca y vuelve a la vista de editar el perfil con la nueva imagen cargada.
  20. El usuario pulsa guardar perfil
  21. El sistema comprueba que todos los campos son válidos.
  22. El sistema guarda el perfil y muestra un mensaje informando de que se ha actualizado correctamente
- **Escenario alternativo:**

  3.a) El usuario no modifica su nombre de usuario.

  4.a) El sistema comprueba que el tamaño del usuario es menor a 5 caracteres.

  4.a.1) El sistema muestra un mensaje en rojo, encima del campo de usuario, indicando que el usuario debe tener al menos 5 caracteres.

5.a) El usuario no modifica su correo electrónico.
6.a) El sistema comprueba que el formato del correo es incorrecto.
7.a.1) El sistema muestra un mensaje en rojo, encima del campo de correo, indicando que el correo tiene un formato incorrecto.
7.a) El usuario no modifica su nombre.
8.a) El usuario no modifica sus apellidos.
9.a) El usuario no modifica su fecha de nacimiento.
10.a) El sistema comprueba que la fecha tiene un formato incorrecto.
10.a.1) El sistema restaura el campo, con el último valor válido.
11.a) El usuario no modifica su sexo.
12.a) El sistema comprueba que el sexo introducido es inválido.
12.a.1) El sistema restaura el campo, con el último valor válido.
13.a) El usuario no modifica su país.
14.a) El sistema comprueba que el país introducido es inválido.
15.a.1) El sistema restaura el campo, con el último valor válido.
15.a) El usuario no modifica el idioma de su perfil.
16.a) El usuario no pulsa el botón de modificar avatar.
18.a) El usuario cierra la biblioteca.
18.a.1) El sistema vuelva a mostrar la vista de editar perfil.
20.a) El usuario pulsa el botón para volver a la vista de perfil.
20.a.1) El sistema descarta los cambios y muestra la vista del perfil.
21.a.1) El sistema comprueba que hay campos que son inválidos.
21.a.1.1) El sistema comprueba que el nombre de usuario, ya está utilizado por otro usuario registrado.
21.a.1.1.1) El sistema muestra un mensaje de error, indicando que el nombre de usuario ya está utilizado.
21.a.1.2) El sistema comprueba que el correo ya está usado por otro usuario.
21.a.1.2.1) El sistema muestra un mensaje de error, indicando que el correo electrónico ya está utilizado.
21.a.2) El sistema no tiene conexión con el servidor.
21.a.2.1) El sistema muestra un mensaje de error, indicando que se ha perdido la conexión con el servidor.

CU07 - Visualizar la información de una película:

- **Descripción:** El usuario podrá visualizar la información detallada de la película.
- **Pre-condición:** El usuario tiene que estar identificado en el sistema y estar en la vista de clasificación swipe, la vista de lista general o en la vista de una lista concreta.
- **Post-condición:** El usuario está en la vista de detalle de la película seleccionada
- **Prioridad:** media
- **Escenario principal:**
    1. El usuario pulsa sobre la carátula de la película.

2. El sistema muestra la vista de detalle de la película.
3. El sistema carga un listado con todas las puntuaciones actualizadas.
4. El sistema carga un listado con todos los participantes de la película.
5. El sistema carga la información básica de la película: título, duración, año, géneros, sinopsis, productores.

- **Escenario alternativo:**
  3.a) El sistema no tiene conexión con el servidor.
  3.a.1) El sistema muestra un mensaje de error, indicando que se ha perdido la conexión con el servidor.

CU08 - Consultar las películas donde trabaja una persona:

- **Descripción:** El usuario podrá consultar las películas, donde ha trabajado una persona (actor, director, etc.).
- **Pre-condición:** El usuario tiene que estar identificado en el sistema y estar en la vista de detalle de una película.
- **Post-condición:** El usuario visualiza el listado de películas donde ha trabajado el participante seleccionado.
- **Prioridad:** media-baja
- **Escenario principal:**
  1. El usuario pulsa sobre un participante.
  2. El sistema carga la lista de películas donde ha trabajado y muestra la vista.
- **Escenario alternativo:**
  2.a) El sistema no tiene conexión con el servidor.
  2.a.1) El sistema muestra un mensaje de error, indicando que se ha perdido la conexión con el servidor.

CU09 - Búsqueda de películas:

- **Descripción:** El usuario podrá buscar una película por el título.
- **Pre-condición:** El usuario tiene que estar identificado en el sistema y estar en la vista clasificación swipe o en la vista de listas general.
- **Post-condición:** El usuario visualiza el listado de películas contenga la palabra búsqueda en su título.
- **Prioridad:** media
- **Escenario principal:**
  1. El usuario pulsa el botón de búsqueda.
  2. El sistema muestra la vista de búsqueda.
  3. El usuario introduce la palabra en el campo del buscador y pulsa el botón buscar.
  4. El sistema realiza una búsqueda por el título original y por el título en el idioma del perfil.
  5. El sistema cargara el listado con los resultados de la búsqueda.
- **Escenario alternativo:**
  3.a) El usuario le da al botón de volver.

3.a.1) El sistema muestra la vista desde donde se accedió a la búsqueda.

4.a) El sistema no tiene conexión con el servidor.

4.a.1) El sistema muestra un mensaje de error, indicando que se ha perdido la conexión con el servidor.

CU10 - Visualizar lista completa:

- **Descripción:** El usuario podrá cargar un listado con todas las películas que estén en una lista.
- **Pre-condición:** El usuario tiene que estar identificado en el sistema y estar en la vista de listas general.
- **Post-condición:** El usuario visualiza el listado de películas de la lista elegida.
- **Prioridad:** media
- **Escenario principal:**
  1. El usuario pulsa el botón con un más de una lista.
  2. El sistema cargará el listado con las películas de la lista elegida, ordenado por fecha de la clasificación (primero las últimas películas clasificadas) y mostrará la vista.
- **Escenario alternativo:**

  2.a) El sistema no tiene conexión con el servidor.

  2.a.1) El sistema muestra un mensaje de error, indicando que se ha perdido la conexión con el servidor.

4.2. Arquitectura

Hemos seguido los principios de una arquitectura REST, que básicamente es una arquitectura cliente/servidor con una serie de particularidades. Entre ellas, podemos destacar las dos siguientes: no guarda el estado y utiliza servicios para las comunicaciones a través del protocolo HTTP. Esto reduce los costes de mantenimiento del sistema ya que, al ser cada parte más independiente, la modificación y la corrección de errores es menos costosa. En la Figura 4.2 podemos ver el esquema que ilustra la arquitectura utilizada.

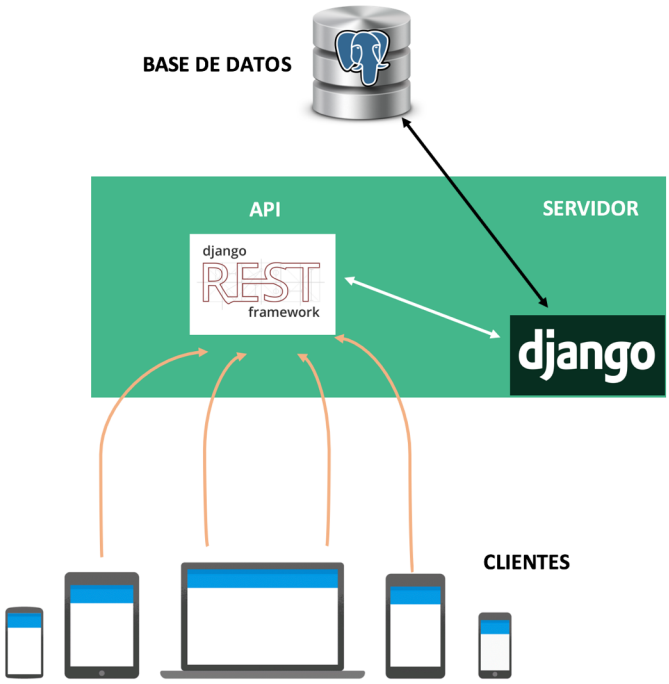

Figura 4.2 Arquitectura REST

A la hora de diseñar las aplicaciones móviles, hemos seguido el patrón arquitectónico Modelo-Vista-Controlador, que separa los datos, la lógica de negocios y la interfaz de usuario. Esta separación de conceptos favorece la reutilización de código, facilita el desarrollo de la aplicación y su mantenimiento.

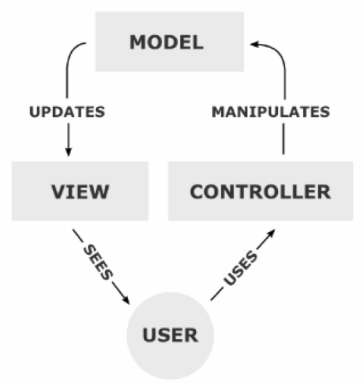

Figura 4.3 Patrón Modelo Vista Controlador (Wikipedia®, 2016)

4.3. Base de datos

Para la base de datos hemos optado por una solución relacional, ya que la información que guardamos, referente a las películas, tiene una gran cantidad de relaciones entre sí; además, así nos aseguramos de evitar duplicidades de la misma.

En concreto hemos optados por el Sistema Gestor de Bases de Datos PostgreSQL que, como hemos comentado en el capítulo sobre herramientas y tecnologías utilizadas, es relacional. Adicionalmente, otra de sus características más destacadas es que es el sistema de código abierto más potente y que, además, encaja perfectamente con Django, el framework utilizado en el desarrollo del servidor.

A continuación se describirá la estructura de la base de datos. Hemos separado la estructura en dos diagramas de entidad-relación para facilitar la legibilidad. Podemos observar en la Figura 5.1 y Figura 5.2 ambos diagramas relacionados a través de las relaciones Collection-Movie y Profile-Celebrity.

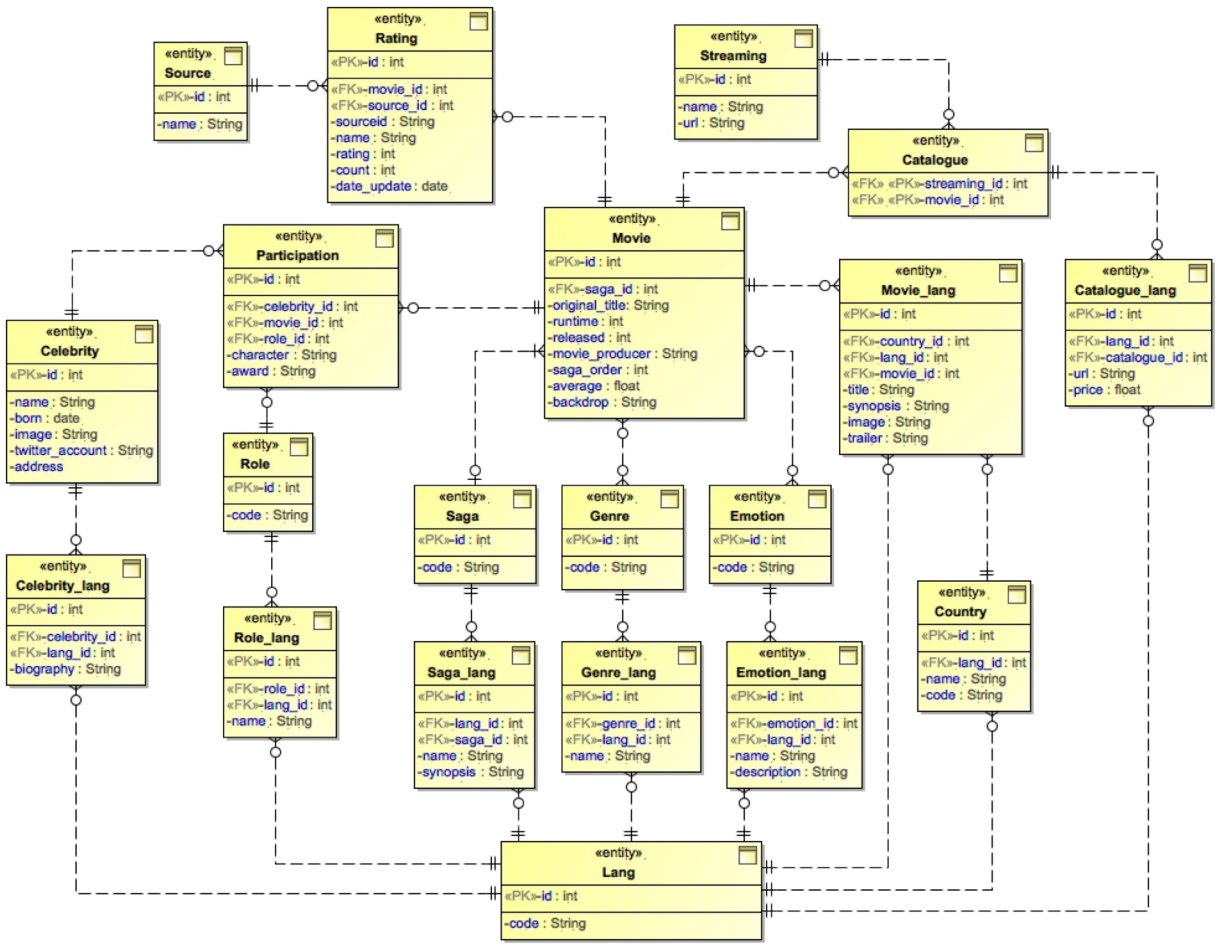

Figura 5.1 Diagrama Entidad-Relación, datos de las películas.

El diagrama entidad-relación de la figura 5.1 representa la estructura de los datos guardados sobre las películas. Las principales entidades son:

- **Movie**: Contiene la información de la película, serie, etc.
- **Celebrity**: Contiene la información de los actores, directores, etc.
- **Source**: Contiene información de los sitios de donde se obtienen las puntuaciones de las películas.
- **Rating**: Esta entidad se genera por la relación mucho a muchos entre “Movie” y “Source”; en ella se guardan las puntuaciones, la fecha de actualización “date_update” e información necesaria para actualizar la puntación.
- **Participation**: Esta entidad se genera por la relación muchos a muchos entre “Movie” y “Celebrity”; en ella guardamos información referente a la participación en la película, su función (actor, director, etc.), premios y nombre del personaje en la película, si es un actor.
- **Lang**: Contiene el código de los lenguajes soportados.

Para cumplir con el requisito funcional RF11, se ha creado una estructura de base de datos que, sin incurrir en duplicidades, proporciona un soporte multilenguaje, preparado para incorporar nuevos lenguajes con facilidad, sin necesidad de modificar la estructura. Las entidades acabadas en “_lang”, surgen de la relación muchos a muchos entre “Lang” y otras entidades que posean datos sensibles de traducción. Estas contendrán la información traducida en cada idioma soportado.

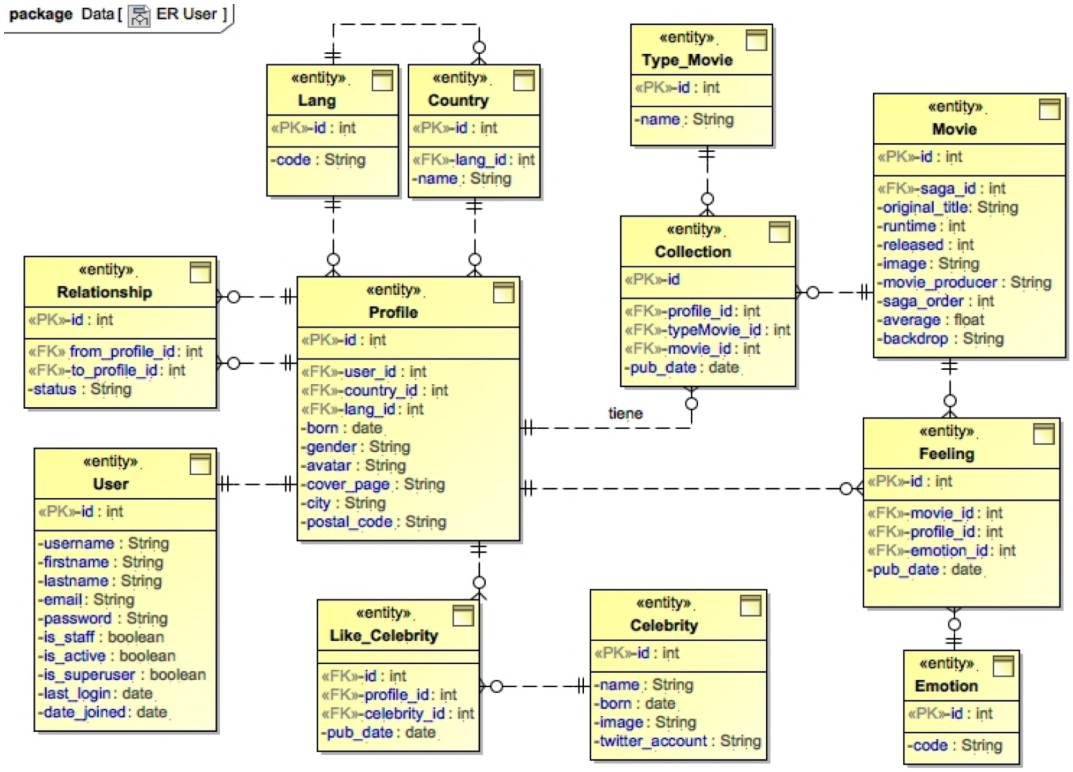

Figura 5.2 Diagrama de Entidad-Relación, datos usuarios.

El diagrama de entidad-relación de la figura 5.2 representa la estructura de los datos de los usuarios registrados. Las principales entidades son:

- **User**: Contiene los relacionados directamente con el registro, autenticación e información básica del usuario.

- **Profile**: Contiene los datos del perfil de usuario, que complementan a los datos de “User”.
- **Collection**: Contiene las películas clasificadas por el usuario.
- **Feeling**: Contiene las emociones que te ha hecho sentir la película.
- **Like_Celebrity**: Contiene los actores, directores, etc., favoritos del usuario.
- **Relationship**: Contiene el conjunto de los usuarios que le siguen y el conjunto de los usuarios a los cuales sigue. Esta entidad se genera por la relación muchos a muchos entre “Profile” y sí misma, con la restricción de que es una relación asimétrica, es decir, un usuario podrá seguir a otro, pero este otro no tiene por qué seguir al primero.

Como podemos observar en la figura 5.2, la entidad “Profile” está relacionada con la entidad “Lang”. En este caso se utiliza para la configurar el idioma del perfil, los datos de las películas se mostrarán en función de esa configuración.

Capítulo 5. Implementación e Instalación

5.1. API REST

Como hemos comentado en el apartado 4.2, hemos utilizado una arquitectura REST. A continuación, mostraremos los servicios ofrecidos por la API. Hemos separado los servicios en dos bloques: “User”, donde detallaremos los servicios relacionados directamente con el usuario y “Movie”, donde detallaremos los relacionados con las películas, series, etc.

Normalmente las llamadas a todos los servicios tendrán una cabecera con dos parámetros básicos: “Authorization” que lo llevaran todas las llamadas que no sean de tipo AUTH, y "Content-Type", que podrá ser de tipo "application/json” o “application/x-www-form-urlencoded”.

5.1.1. User

- **AUTH - PostLogin:** Iniciar sesión en el sistema.

  `(POST) /users/login/`

  Datos del formulario

Campo	Tipo	Descripción
username	*Text*	Nombre de usuario o email registrado en el sistema
password	*Text*	Contraseña asociada al usuario

  Respuesta

```
{
    "message":"Login successfully",
    "status":200,
    "user":{
            "username":"movie",
            "profile": {
                    "avatar":"/media/user/default/no-image.png",
                    "lang": {
                            "code":"es"
                    }
            },
            "id":9,
            "email":"movie@mooviest.com"
    },
    "token":"8a0ac632536d7d1ac5db90f6f05338cef2778516"
}
```

  Obtenemos los datos del usuario y su perfil, además del “*token”* necesario para realizar el resto de peticiones que no son de tipo *AUTH* (Autorizadas).

Error 4xx


```
{
    "message": "User or password incorrect",
    "status": 404,
    "user": null,
    "token": null
}
```


Si el usuario o la contraseña no coinciden, devolverá un mensaje de error.

- **AUTH - PostSignUp:** Registrar un usuario en el sistema.
  `(POST) /users/`

  Datos del formulario

Campo	Tipo	Descripción
username	*Text*	Nombre de usuario
email	*Text*	Email
password	*Text*	Contraseña
profile.lang.code	*Text*	Código de idioma. Actualmente los valores posibles son: *es*: español, *en*: inglés

  Respuesta


```
{
      "status": 201,
      "user": {
            "id": 9,
            "username": "movie",
            "email": "movie@mooviest.com",
            "profile": {
                  "lang": {
                        "code": "es"
            },
           "avatar": "/media/user/default/no-image.png"
            }
      },
      "errors": null,
      "token": "8a0ac632536d7d1ac5db90f6f05338cef2778516"
  }
```


  Obtenemos los datos del usuario y su perfil, además del *token*, necesario para realizar el resto de peticiones que no son de tipo *AUTH* (Autorizadas).

Error


```
{
    "status": 400,
    "user": null,
    "errors": {
          "username": [
                "Ya existe un usuario con ese nombre de usuario."
          ],
          "email": [
                "Ya existe un usuario con ese email."
          ]
    },
    "token": null
}
```


Si ya existe un usuario en el sistema con los campos introducidos nos devolverá un campo de error con los mensajes correspondientes.

- **User - GetUserProfile:** Obtener el perfil de un usuario.
  `(GET) /users/{id}/`

Datos en la URL

Campo	Tipo	Descripción
id	*Number*	Id de un usuario registrado en el sistema

Obtenemos todos los datos del usuario y su perfil.

Respuesta


```
{
    "status": 200,
    "user": {
    "first_name": "",
    "last_name": "",
    "username": "movie",
          "profile": {
                "born": null,
                "gender": null,
                "avatar": "/media/user/default/no-image.png",
                "lang": {
                      "code": "es"
                },
                "postalCode": null,
                "city": null
          },
          "id": 9,
          "email": "movie@mooviest.com"
    }
}
```

- **User - UpdateUserProfile:** Actualizar el perfil de un usuario.
  `(PUT) /users/{id}/`

  Datos en la URL

Campo	Tipo	Descripción
id	*Number*	Id de un usuario registrado en el sistema

  Datos del formulario

Campo	Tipo	Descripción
username	*Text*	Nombre de usuario
profile.lang.code	*Text*	Código de idioma
first_name (opcional)	*Text*	Nombre
last_name (opcional)	*Text*	Apellidos
email (opcional)	*Text*	Email
profile.born (opcional)	*Date*	Fecha de nacimiento
profile.city (opcional)	*Text*	Ciudad
profile.postalCode (opcional)	*Text*	Código postal
image (opcional)	*Image*	Imagen del perfil

  Respuesta
  Devuelve la misma respuesta que en la petición **GetUserProfile.**

  User - GetSwipeList: Obtener la lista de películas para el sistema de clasificación swipe, es decir, películas que no están clasificadas en ninguna lista del usuario.
  `(GET) /users/{id}/swipelist/`

  Datos en la URL

Campo	Tipo	Descripción
id	*Number*	Id de un usuario registrado en el sistema

Respuesta


```
{
  "count": 10,
  "next": null,
  "previous": null,
  "results": [
     {
     "movie_lang_id": 54625,
     "image": "pelis/HP9YTUHXX6",
     "backdrop": "pelis/HP9YTUHXX6",
     "title": "Cartas desde la locura",
     "collection": null,
     "id": 28689,
     "average": 0
    },…
  ]
}
```


Obtenemos la información necesaria para mostrar en el sistema de clasificación swipe.

- **User - GetCollectionList:** Obtener la lista de películas de un tipo concreto.
  `(GET) /users/{id}/collection/?name={name}&page={page}`

Datos en la URL

Campo	Tipo	Descripción
id	*Number*	Id de un usuario registrado en el sistema
name	*Text*	Nombre de la colección. Los tipos de colección son: seen, favourite, watchlist, blacklist
Page	*Number*	Número de página

Respuesta


```
{
    "count": 174,
    "next": "/users/9/collection/?name=favourite&page=2",
    "previous": null,
    "results": [
          {
          "movie_lang_id": 2919,
          "image": "/a4/9d/a49dd25d5df38f2e97c4bfd7d4875684.jpg",
          "backdrop": "/a4/9d/a49dd25d5df38f2e97c4bfd7d4875684.jpg",
          "title": "El Señor de los Anillos: La Comunidad del Anillo",
          "collection": {
                "typeMovie": "favourite",
                "id": 1
          },
          "id": 1468,
          "average": 0
          },…
    ]
}
```


Obtenemos un resultado paginado con la información necesaria para mostrar las películas en la vista de listas general y en la de la lista concreta.

- **AUTH - PostMovieCollection:** Clasificar una película en la colección de un usuario determinado.
  `(POST) /collection/`

  Datos del formulario

Campo	Tipo	Descripción
user	*Number*	Id de un usuario del sistema
movie	*Number*	Id de una película del sistema
typeMovie	*Text*	Nombre de la colección. Los tipos de colección son: seen, favourite, watchlist, blacklist

  Respuesta

```
{
      "id": 854,
      "user": 9,
      "movie": 3088,
      "typeMovie": "favourite",
      "pub_date": "2016-11-06T07:44:48.756881Z"
}
```

  Obtenemos el resultado de haber clasificado una película en una colección determinada; la id de la tabla “Collection”, la del usuario, la de la película, el tipo de la colección y la fecha de clasificación.

- **AUTH - PatchMovieCollection:** Actualizar la colección de una película de un usuario.
  `(PATCH) /collection/{id}/`

  Datos en la URL

Campo	Tipo	Descripción
id	*Number*	Id de la tabla ***Collection*** donde está clasificada la película que queremos actualizar

  Datos del formulario

Campo	Tipo	Descripción
typeMovie	*Text*	Nombre de la colección. Los tipos de colección son: seen, favourite, watchlist, blacklist

  Respuesta

```
{
   "id": 854,
   "user": 9,
   "movie": 3088,
   "typeMovie": "seen",
   "pub_date": "2016-11-06T07:48:47.107361Z"
}
```

  Obtenemos el resultado de haber actualizado el tipo de colección de una película; la id de la tabla “Collection”, la del usuario, la de la película, el tipo de la colección y la fecha de clasificación.

5.1.2. Movie

- **Movie - GetMovieDetail:** Obtener la información detallada de una película.
  `(GET) /movie/{id}/?movie_lang_id={_}&user_id={_}`

  Datos en la URL

Campo	Tipo	Descripción
id	*Number*	Id de la película a obtener
movie_lang_id	*Number*	Id de la tabla “Movie_lang” que identifica a la película en un idioma.
user_id	*Number*	Id del usuario que obtiene la película.

  Respuesta


```
{
      "title": "Piratas del Caribe: La maldició …",
      "participations": [
        {
          "celebrity": {
            "id": 6687,
            "name": "Terry Rossio",
            "born": null,
            "image": "/0c/c8/0cc8bb88f9135835849485bf8f27cf7b.jpg",
            "twitter_account": "",
            "address": ""
          },
          "role": "Escritor",
          "character": "",
          "award": ""
        },…
      ],
      "collection": {
          "typeMovie": "seen",
          "id": 854
      },
      "movie_lang_id": 6140,
      "movie_producer": "Walt Disney Pictures … ",
      "synopsis": "El aventurero Capitán Jack...",
      "id": 3088,
      "genres": [{"name": "Fantasía"}, {"name": "Acción"},
          { "name": "Aventura"}
      ],
      "runtime": 143,
      "original_title": "Pirates of the Caribbean: The …",
      "released": 2003,
      "average": 0,
      "country": null,
      "backdrop": "/1d/8f/1d8f6066a9ad1cb91211bcbfca5915f2.jpg",
      "image": "/1d/8f/1d8f6066a9ad1cb91211bcbfca5915f2.jpg"
    "ratings": [
        {
          "name": "IMDb",
          "rating": 81,
          "count": 811554,…
```


Obtenemos la información detallada de una película.

- **Movie - GetSearchMovies:** Obtener el resultado de buscar películas por su título original o por el del idioma del usuario que realiza la búsqueda.
  `(GET) /movie_lang/?title={title}&code={code}&page={page}`

  URL

Campo	Tipo	Descripción
title	*Text*	Título de una película a buscar.
code	*Text*	Código de idioma. Actualmente los valores posibles son: *es*: español, *en*: inglés
page	*Text*	Número de página

  Respuesta


```
{
  "count": 6,
  "next": null,
  "previous": null,
  "results": [
    {
      "title": "Piratas del Caribe: El cofre del hombre muerto",
      "average": 0,
      "collection": null,
      "movie_lang_id": 6136,
      "image": "/1c/3e/1c3e15050ceaf0e4edb15cdfb3c35867.jpg",
      "id": 3086
    },…
  ]
}
```


  Obtenemos un resultado paginado con la información necesaria para mostrar las películas en la vista lista concreta.

5.2. Scripts

Como comentamos en el apartado 3.1, unas de las preocupaciones que teníamos era, la recopilación de información, con el objetivo de proporcionar una base de datos completa y poder ofrecer así un buen servicio.

La finalidad de estos scripts, es recopilar todos los datos de las películas, además de extraer todas las puntuaciones de los diferentes sitios que proporcionan información de este tipo. Los scripts se han estructurado e implementado pensado, además, de en la recopilación previa de datos, en su futura utilización para añadir nuevas películas a la base de datos y mantener actualizadas las puntuaciones.

Se han estructurado en función del origen de los datos, del sistema utilizado para obtenerlos y el tipo de datos extraído. Se ha creado un script principal "main_script.py", que es el encargado de ejecutar todos los demás en el momento preciso. Este script introduce los datos a través de la API comentada en el apartado anterior, usando los métodos de "interface_db" para esta tarea.

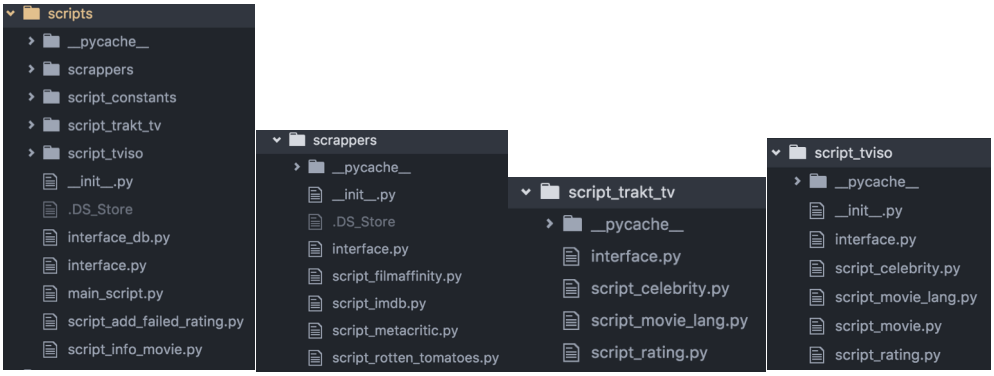

Figura 5.3 Estructura de los scripts

Además de controlar los posibles errores, utilizando los métodos de "interface.py" del directorio raíz scripts, los guardará en un log y si se interrumpe la ejecución del script, se enviará un correo con el "log" indicando el motivo de la finalización.

Podemos afirmar que el sitio principal de extracción de datos ha sido "Tviso". Además de extraer su puntuación hemos obtenido, mediante su API, todos los datos de las películas en español. De "Trakt_tv", hemos obtenido la puntuación y los datos de las películas en inglés, carátula incluida. Los "scrappers" se han utilizado para extraer la puntuación de los sitios que no tenían API.

A continuación, pondremos dos ejemplos de extracción de puntuación: uno mediante API y otro mediante un scrapper:

- Extracción mediante Api (Trakt_tv):

```
def insert_rating(db, movie_id, imdb_id):
    error_code = False
    error_message = ""
    url_movie = "/movies/" + imdb_id + "?extended=full"

    data = interface.get_info(url_movie)
    try:
        sourceid = int(data["ids"]["trakt"])
        rating = int(data["rating"] * 10)
        count = int(data["votes"])
   …
```

Llamamos al método con la “id” de la película de nuestra base de datos y la de, en este caso, “imdb”, aunque también puede ser la id de “trakt_tv”. Con estos datos podremos obtener fácilmente los datos de la puntuación, para insertarla en la base de datos. Posteriormente, cuando queramos actualizar la base de datos, solo tendremos que llamar al script correspondiente, pasándole el parámetro “source_id” que tenemos guardado en la tabla Rating.

- Extracción mediante scrapper (IMDb):

```
from bs4 import BeautifulSoup
…
        rating = soup.find(itemprop="ratingValue").get_text().strip()
        count = soup.find(itemprop="ratingCount").get_text().strip()
        rating = int(rating.replace(".", ""))
        count = int(count.replace(",", ""))
 …
```

Para implementar los scrappers hemos utilizado la librería “BeautifulSoup” (Richardson, 2004-2015). En el fragmento del código anterior, podemos ver la gran diferencia con el script utilizando API. La llamada al script sería la misma, pero en el scrapper existe una función aparte para extraer los datos.

5.3. Creación e instalación del proyecto.

El proceso se puede separar en dos partes: la del servidor y la del cliente, en este caso, la aplicación IOS. En ellas explicaremos cómo crear y configurar el proyecto, a partir de la carpeta del proyecto o de nuestro repositorio de GitHub.

5.3.1. Servidor Django

Para comenzar, procederemos a instalar un gestor de paquetes: en el caso de que utilicemos Linux nos valdrá con “apt-get” que viene por defecto. En Mac hay varias opciones, pero una de las más utilizadas es Homebrew. Para instalarlo, visitamos su sitio web y seguimos las instrucciones de instalación. En el caso de Windows habrá descargar los instaladores de las aplicaciones utilizadas.

A partir de aquí la instalación es parecida para todas las plataformas; en principio nos hará falta instalar el servidor de base de datos PostgreSQL:

```
# macOS
$ sudo brew install postgresql
# Linux
$ sudo apt-get install postgresql
```

Después creamos la base de datos de la aplicación:

```
# Create the database (as root)
$ createdb mooviest
$ createuser -P
# Activate the PostrgreSQL CLI to grant privileges to
the user root
$ psql

>> GRANT ALL PRIVILEGES ON DATABASE mooviest TO root;
```

Con el último comando entramos a la consola de postgresql y, desde ahí, le daremos todos los privilegios de la base de datos al usuario root. Una vez instalado el servidor de base de datos, el próximo paso es instalar git, para clonar el repositorio:

```
# macOS
$ sudo brew install git
# Linux
$ sudo apt-get install git
```

Podremos copiar la carpeta del proyecto en cualquier sitio; en el caso de utilizar el repositorio de GitHub, lo clonaremos con el siguiente comando:

```
$ git clone https://github.com/JoseAntpr/mooviest.git
```

Para el proyecto se utiliza Python 3; si no lo tenemos instalado, tendremos que hacerlo con los siguientes comandos:

```
# macOS
$ sudo brew install python3
# Linux
$ sudo apt-get install python3
```

Posteriormente, procederemos a instalar las dependencias: entramos a la carpeta del proyecto, “requirements.txt”, que contiene todas las dependencias necesarias, así que solo tendremos que ejecutar el siguiente comando:

```
pip3 install –r requirements.txt
```

Pasamos a configurar la base de datos. Para ello accedemos al archivo “settings_develop.py” y completaremos la información con los datos necesarios para la creación de la base de datos. El resultado sería el siguiente:

```
DATABASES = {
    'default': {
        'ENGINE':
'django.db.backends.postgresql_psycopg2',
        'NAME': 'mooviest',
        'USER': 'root',
        'PASSWORD': 'root',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

Ahora crearemos las migraciones y las ejecutaremos para generar la estructura de la base de datos, con los siguientes comandos:

```
python3 manage.py makemigrations
python3 manage.py migrate
```

Ya tenemos todo configurado y listo. Actualmente la base de datos está vacía, así que lo último será cargar todos los datos de las películas. Para ello se ha proporcionado un script denominado “backup_sql.sql”, que automatiza esta tarea con el siguiente comando:

```
psql –U postgres –d mooviest < backup_sql.sql > mooviest.log 2>&1
```

Solo aclarar que “postgres” será el usuario de la base de datos al que le dimos permisos; “mooviest” es el nombre de la base de datos; “backup_sql.sql” será el script proporcionado y “mooviest.log”, el archivo donde se guarde el resultado de importar los datos.

Ya solo nos queda iniciar el proyecto con el siguiente comando:

```
python3 manage.py runserver
```

Con este comando solo se ejecutará en local, por lo tanto, solo funcionará desde el mismo equipo que se ejecute, en el puerto 8000 por defecto. Si añadimos 0.0.0.0: Puerto, será accesible desde cualquier máquina de la red.

5.3.2. Aplicación móvil IOS

Tendremos que tener un mac, instalar Xcode 8.1 y para las dependencias del proyecto instalaremos el gestor de paquetes Carthage, con el comando:

```
$ brew install carthage
```

Podremos copiar la carpeta del proyecto en cualquier sitio; en el caso de utilizar el repositorio de GitHub, lo clonaremos con el siguiente comando:

```
$ git clone https://github.com/JoseAntpr/mooviest_ios.git
```

Para instalar las dependencias, entramos en la carpeta del proyecto. El archivo "Cartfile" contiene todas las dependencias necesarias, así que solo tendremos que ejecutar el siguiente comando:

```
$ carthage update --platform IOS
```

Una vez finalizado, solo tenemos que abrir el proyecto y ejecutarlo, también tendremos que tener iniciado el servidor para que funcione.

5.4. Desarrollo y manual aplicación móvil IOS

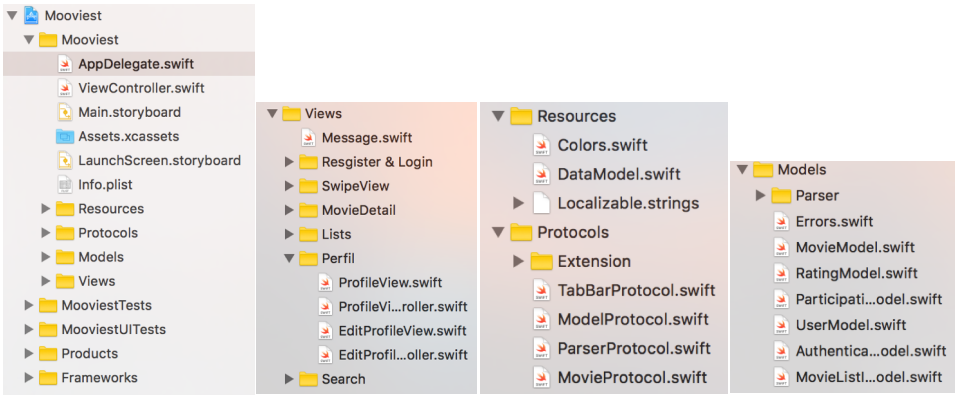

Figura 5.4 Estructura aplicación móvil IOS

Comenzaremos explicando cómo se ha estructurado el proyecto y describiremos los archivos más importantes:

- **AppDelegate.swift:** Es el punto inicial desde donde parte la aplicación, similar a la clase "Application" en Android. En él se recogen una serie de eventos de la aplicación, tales como la terminación de la carga de opciones, cuando se cierra la aplicación etc.
- **Info.plist:** Este es el archivo de configuración del proyecto.
- **LaunchScreen.storyboard:** Es la imagen que se verá nada más abrir la aplicación.

Se han separado por un lado las vistas, con su correspondiente controlador, los recursos, donde podemos encontrar los métodos que llaman a la API REST del servidor, y los modelos.

En los siguientes subapartados mostraremos las diferentes vistas de la aplicación, explicaremos su funcionamiento y enseñaremos las partes de código más reseñables.

5.4.1. LaunchScreen, inicio de sesión y registro

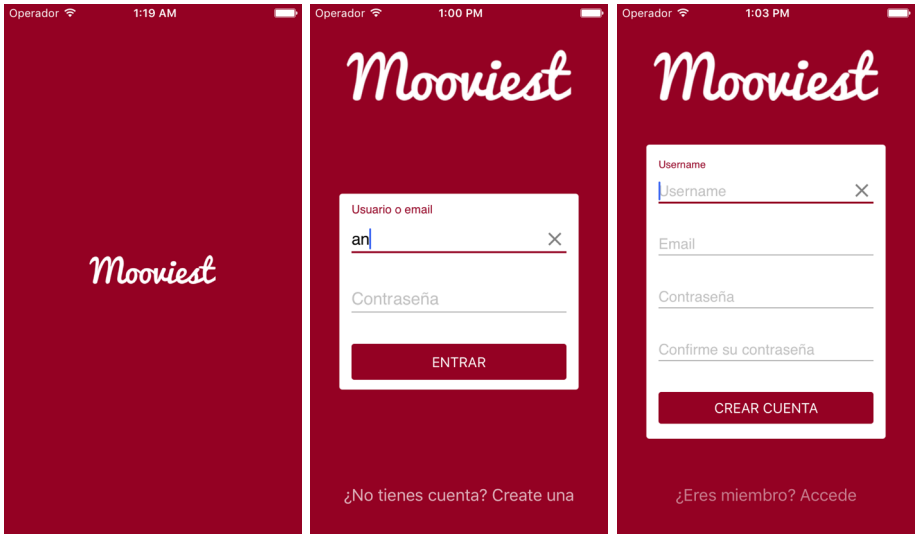

Figura 5.5 LaunchScreen, vista de inicio de sesión y registro

La primera vista es el LaunchScreen, comentado anteriormente. Esta vista se muestra mientras se está cargando la aplicación.

La siguiente vista es la de inicio de sesión. Si ya tenemos cuenta en la aplicación, introducimos el usuario o email y la contraseña, accediendo al sistema. Si alguno de los parámetros es incorrecto, se muestra un mensaje de error; si por alguna razón no tenemos conexión con el servidor, también se mostrará un mensaje de error.

Si por el contrario no estamos registrados, pulsando el texto de la parte de abajo, aparecerá el formulario de inicio con una animación de izquierda a derecha. Rellenamos los campos, que tienen asignados su validadores y, si todo es correcto, se creará la cuenta y se iniciará la sesión. En caso contrario, se mostrará el mensaje de error.

Como explicamos en el apartado de la API, la llamada para iniciar sesión y la de registro nos devuelve la misma respuesta: el usuario y su token asociado; quizás la serialización de estos datos sería lo más destacable de este apartado.

```
//Extract token
guard let token = json["token"] as? String else{
    throw SerializationError.missing("token")
}
//Extract user
guard let userJson = json["user"] as? [String:Any] else {
    throw SerializationError.missing("user")
}
```

Las guardas (guard), que se introdujeron en Swift 2, nos permiten realizar chequeos en el flujo de nuestro programa. Si esta no es válida, lanzaremos una excepción que indicará dónde ha fallado la serialización. Se ha seguido la guía que aparece en la página oficial de Apple, “Working with JSON in Swift”, para la creación de los serializadores.

En referencia al diseño, en este apartado, se ha mantenido el mismo estilo que en la aplicación desarrollada en Android, para que hubiera una uniformidad entre ambas plataformas móviles. En los siguientes apartados se han seguido las guías de estilo de IOS, manteniendo el color corporativo, pero intentando diseñar una interfaz más limpia y menos saturada.

5.4.2. Clasificación swipe.

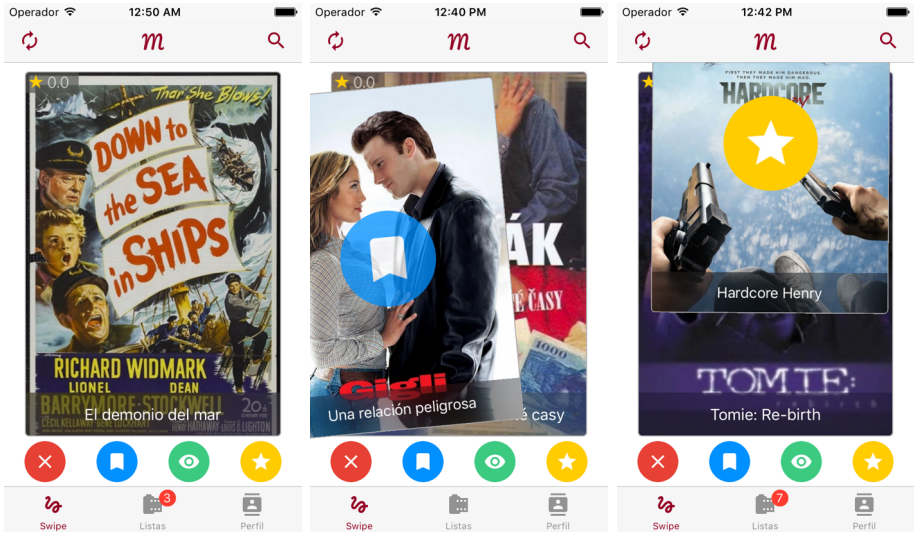

Figura 5.5 Clasificación swipe

La clasificación swipe es la vista que se cargará inmediatamente después de iniciarse la sesión. Es una de las vistas más importante, ya que pretende ser un sistema de clasificación rápido e intuitivo. Se podrá clasificar la película de dos formas:

- Mediante los botones inferiores, que de izquierda a derecha serían: no deseada (cruz), pendiente (marca páginas), vista (ojo) y favorita (estrella).
- Mediante swipe: deslizando la portada hacia abajo “no deseada”, hacia arriba “favorita”, hacia la derecha “vista” y hacia la izquierda “pendiente”.

Cada vez que clasifiquemos una película, se incrementará el número de la pestaña de las listas, para indicar al usuario cuántas películas ha clasificado.

Esta vista quizás sea la más compleja, ya que entran en juego varios elementos: consta de una vista general donde están los botones, la vista de la portada que es deslizable y el icono que aparece encima de la portada, dependiendo de cuál sea la

dirección al deslizar. Se han realizado animaciones, utilizando “transform”, equivalentes a los utilizados en los juegos, para dar movimiento, escalar la vista y realizar la rotación cuando deslizamos la portada.

A continuación, mostraremos los fragmentos de código con lo más destacable:

```
let rotationStrength: Float = min(xFromCenter/ROTATION_STRENGTH, ROTATION_MAX)
let rotationAngle = ROTATION_ANGLE * rotationStrength
var scale = max(1 - fabsf(rotationStrength) / SCALE_STRENGTH, SCALE_MAX)

let rotationStrength2: Float = min(yFromCenter/ROTATION_STRENGTH, ROTATION_MAX)
let scale2 = max(1 - fabsf(rotationStrength2) / SCALE_STRENGTH, SCALE_MAX)
scale = min(scale, scale2)

let translation = CGAffineTransform(translationX:  CGFloat(xFromCenter),
y:CGFloat(yFromCenter))
let rotation = CGAffineTransform(rotationAngle: CGFloat(rotationAngle))
let scaleTransform = rotation.concatenating(translation).scaledBy(x:CGFloat(scale), y:
CGFloat(scale))

self.transform = scaleTransform
self.updateOverlay(sinceXCenter: CGFloat(xFromCenter),sinceYCenter: CGFloat(yFromCenter))
```

En el fragmento de código anterior podemos ver cómo se calcula el desplazamiento, la rotación de la portada conforme se va deslizando. Además vamos actualizando el icono que aparece encima, en función de dónde se encuentre la carátula.

Cuando cargamos la vista, se hace una llamada a la API que nos devuelve un número determinado de películas. Sería poco eficiente cargarlas todas en la vista; lo que hacemos es tener dos listas de películas: una con todas y otra con las películas cargadas. Funcionaría de forma similar a un buffer; hemos definido una constante para indicar el tamaño máximo de ese buffer. El mínimo debería ser al menos dos ya que, si no, cuando se deslice la película no habría ninguna detrás.

```
    func afterSwiped() {
        loadedCards.remove(at: 0)
        allCards.remove(at: 0)
        movies.remove(at: 0)

        loadMoreCards()
        if allCards.count == MIN_CARDS {
            self.loadSwipe()
        }
    }
```

Una vez que clasifiquemos la película, se borrará de la vista y se ejecutará la función “afterSwiped”. Esta borrará la película de las listas, cargará más películas, según el tamaño de buffer indicado, y comprobará si seguimos teniendo suficientes películas en la lista general; si no volverá a hacer una nueva petición a la API.

```
func loadMoreCards() {
        var i = loadedCards.count
        while loadedCards.count < MAX_BUFFER_SIZE && i < allCards.count {
            loadedCards.append(allCards[i])
            v.panelSwipeView.insertSubview(loadedCards[i], belowSubview: loadedCards[i-1])
            setupConstraintsSubView(Index: i)
            i += 1
        }
    }
```

La función “loadMoreCards” cargará nuevas películas en la vista si hay cargadas un número menor que el tamaño del buffer.

5.4.3. Listas de películas del usuario.

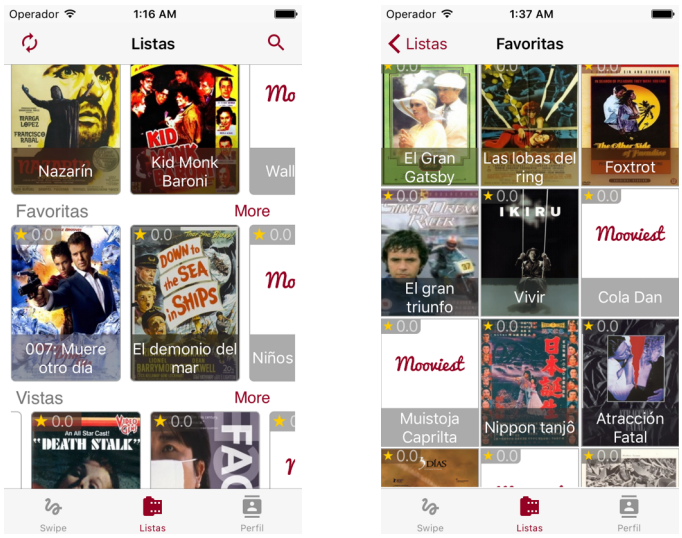

Figura 5.5 Listas de películas de usuario

En la vista de las listas, podremos ver las últimas películas clasificadas ordenada. Si pulsamos el botón “more” de una lista concreta, podremos consultar todas sus películas.

En el apartado de la API, comentamos que la llamada a las listas se hacía con paginación. Quizás la parte más interesante del código es aquella en la que se ha gestionado esta paginación.

```
func collectionView(_ collectionView: UICollectionView,
                        willDisplay cell: UICollectionViewCell,
                        forItemAt indexPath: IndexPath){

        if !isIOS10 {
            if (movies.count-indexPath.row) < MIN_MOVIES {
                nextMovies()
            }
        }
    }
```

La función “collectionView” se ejecutará antes de mostrar una película. Lo que hacemos es que, si quedan un número menor que la constante “MIN_MOVIES”, que

en principio la hemos inicializado a 10, se hará una llamada a la API pidiendo la siguiente página.

A partir de IOS 10 (Apple Inc., 2016) se han introducido una serie de mejoras en las colecciones. Entre ellas la posibilidad de precargar el contenido de la celda para mejorar la respuesta del desplazamiento.

```
func collectionView(_ collectionView: UICollectionView, prefetchItemsAt
indexPaths: [IndexPath]) {
        var urls = [URL]()
        for indexPath in indexPaths {
            let image = movies[indexPath.item].image
            if image != ""{
                urls.append(URL(string: image)!)
            }
        }
        ImagePrefetcher(urls: urls).start()
        if (movies.count - (indexPaths.last?.item)!) < indexPaths.count {
            nextMovies()
        }
    }
```

Aprovechando esta nueva novedad, hacemos una precarga de las portadas de las películas.

5.4.4.Perfil de usuario.

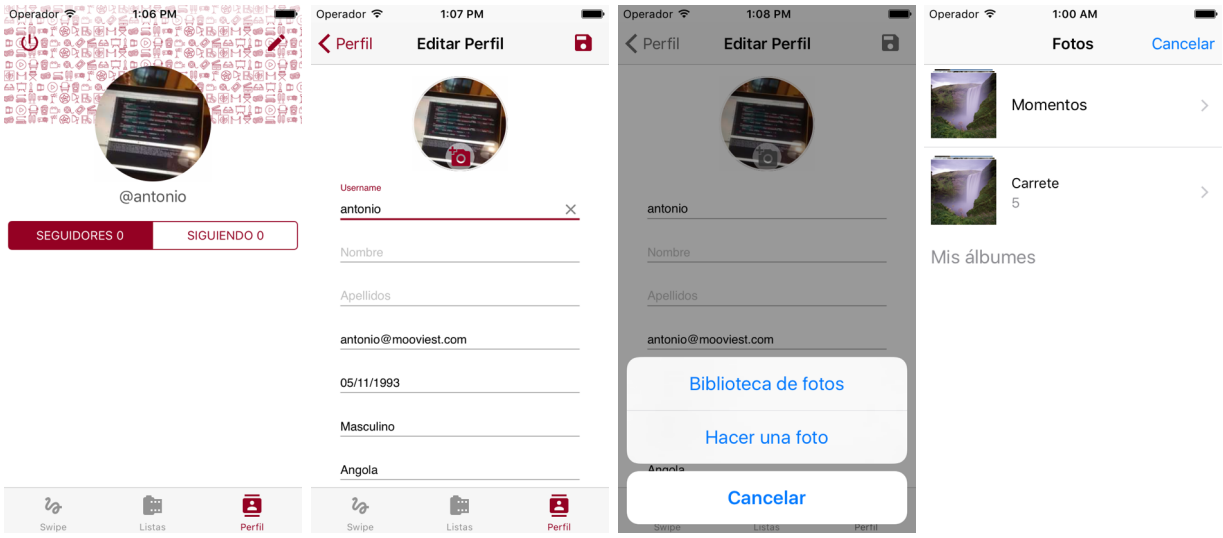

Figura 5.5 Listas de películas de usuario

En la vista de perfil, podemos consultar el nombre de usuario, los seguidores, los usuarios que sigues, la imagen de avatar, etc. Se puede cerrar la sesión pulsando en el botón de arriba a la izquierda y modificar el perfil pulsando el botón de la parte superior derecha.

En la vista de perfil podremos modificar todos los datos del perfil, incluido el avatar y otros datos privados que no se muestran en la vista de perfil.

El código de estas vistas no tiene nada especialmente relevante; quizás podríamos destacar que, para editar la foto y poder acceder a la biblioteca del móvil, se tiene que configurar en el archivo “info.plist” una serie de parámetros.

5.4.5. Búsqueda de películas

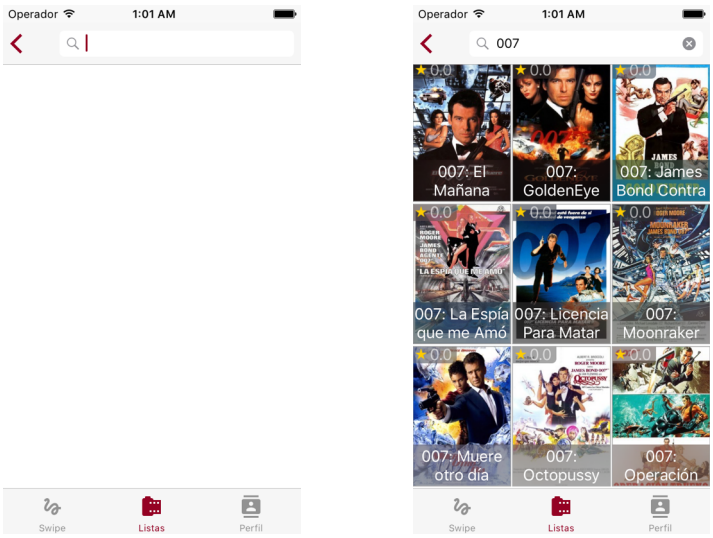

Figura 5.5 Búsqueda de películas

Podremos acceder a esta vista, desde la del swipe o desde la de listas de películas, presionando la lupa situada en la parte superior derecha. Para buscar, simplemente escribimos el nombre de la película o parte del mismo y, al presionar el botón de buscar, cargará los resultados, al igual que la vista de una lista concreta, utiliza el paginado de la misma forma.

5.4.6. Detalle de una película.

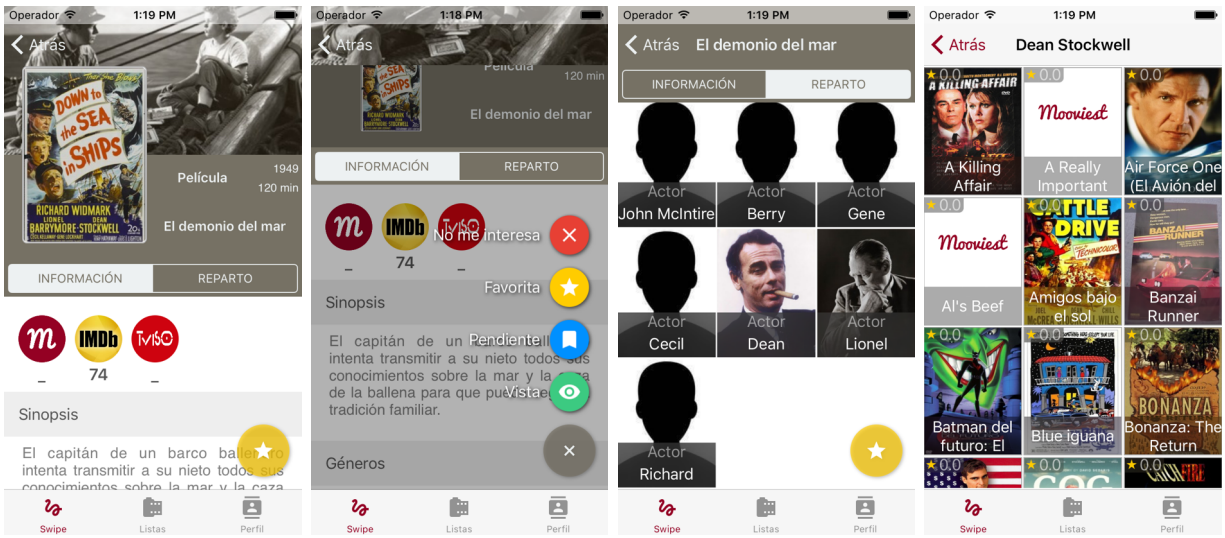

Figura 5.5 Listas de películas

La vista de detalle de la película es otra vista importe. En ella podemos consultar toda la información sobre la película, las puntuaciones, participantes, etc. Si pulsamos en uno de los participantes, podremos ver en qué películas ha trabajado, podremos clasificar la película o cambiar la clasificación, mediante el botón flotante; si ya está clasificada, ese botón tendrá la apariencia de la clasificación elegida.

Podremos acceder a esta vista desde cualquiera que tenga películas, presionando la carátula de la que queramos consultar. Al igual que la vista de swipe, esta contiene muchos elementos que la hace compleja; se utiliza el desplazamiento de "scrolling" para crear la animación de la portada. También se toma el color medio de la carátula o fondo de la película, si tuviera. Este se aplicará a la vista, como resultado tendremos una vista más personalizada y adaptada a la película que consultamos.

```
let headerScaleFactor:CGFloat = -(offset) / v.headerView.bounds.height
let headerSizevariation = ((v.headerView.bounds.height * (1.0 + headerScaleFactor)) -
v.headerView.bounds.height)/2.0
headerTransform = CATransform3DTranslate(headerTransform, 0, headerSizevariation*0.5, 0)
cardTransform = CATransform3DTranslate(headerTransform, 0, headerSizevariation, 0)
headerTransform = CATransform3DScale(headerTransform, 1.0 + headerScaleFactor, 1.0 +
headerScaleFactor, 0)
noScrollTransform = cardTransform
if v.coverView.layer.zPosition < v.headerView.layer.zPosition{
    v.headerView.layer.zPosition = 0
}
```

En esta parte de código actualizamos los "transform", cuando el movimiento del "scrolling" se hace hacía abajo. Tendremos un "transform" para la cabecera, otro para la portada y otro para los elementos que no queramos que se muevan.

```
//Alpha
v.headerBackdropImageView.alpha = max (0, (1-( offset-
offset_BackdropFadeOff*3)/distance_W_LabelHeader)/10)
let offsetTitle  = max(offset_CardProfileStop-offset,0)
self.navigationController?.navigationBar.setTitleVerticalPositionAdjustment(offsetTitle,
for: .default)
let offsetCaTabs  = max(offset-offset_CardProfileStop,0)

v.castCollectionView.contentOffset.y = offsetCaTabs

//Animations
headerTransform = CATransform3DTranslate(headerTransform, 0, max(-offset_HeaderStop, -
offset), 0)
cardTransform = CATransform3DTranslate(cardTransform, 0, max(-offset_CardProfileStop, -
offset), 0)
noScrollTransform = CATransform3DTranslate(noScrollTransform, 0, -offset, 0)

let avatarScaleFactor = (min(offset_CoverStopScale, offset)) / v.coverView.bounds.height
let avatarSizeVariation = ((v.coverView.bounds.height * (1.0 + avatarScaleFactor)) -
v.coverView.bounds.height)
avatarTransform = CATransform3DTranslate(avatarTransform, 0, avatarSizeVariation*0.5, 0)
avatarTransform = CATransform3DScale(avatarTransform, 1.0 - avatarScaleFactor, 1.0 -
avatarScaleFactor, 0)
```

En esta parte del código actualizamos los "transform" cuando se realice el desplazamiento hacia arriba, controlaremos la transparencia del fondo, cuando esté

completamente arriba no será totalmente transparente. Calcularemos el desplazamiento y escalado de la portada de la película, etc.

Para finalizar, aclarar que no se ha utilizado el “storyboard”, que es el editor visual para crear vistas. Todas las vistas están creadas por código y utilizando “autolayout”. Esto quiere decir que todas las vistas se adaptan proporcionalmente al tamaño del teléfono donde se ejecuta. Las animaciones creadas, también se adaptan al tamaño del teléfono

Capítulo 6. Conclusiones y trabajo futuro

6.1. Conclusiones

A lo largo de la memoria, hemos expuesto todos los puntos del proyecto, motivación, objetivos, conocimientos previos, especificación y desarrollo. Antes de dar mi conclusión final, quisiera exponer los problemas más destacables que nos han surgido y su repercusión en el proyecto.

Nunca había utilizado el lenguaje “Python” ni el “framework Django”. Por esta razón, hemos tenido que hacer una serie de tutoriales para iniciarnos en este lenguaje. Durante el desarrollo del servidor hemos tenido que consultar documentación más específica. Como consecuencia, su desarrollo ha sido más lento de lo indicado en la temporización del anteproyecto.

La recopilación de datos ha sido otra de las partes que han conllevado un considerable retraso en el proyecto. Además del tiempo empleado en la comunicación con las diferentes fuentes de donde hemos obtenidos los datos de las películas, el tiempo utilizado en la creación de los scripts que extraen esta información, e incluso los errores detectado en algunas de las “APIs” utilizadas han supuesto retrasos en la planificación inicial. No hemos tenido en cuenta el tiempo de ejecución de estos scripts, ni la limitación del número de peticiones de las diferentes “APIs”. Como consecuencia, decidimos solo extraer la información de las películas, aunque nuestra plataforma está preparada para ofrecer información de todo tipo de contenidos, ahora mismo solo ofrecemos información de películas.

Con respecto a la aplicación IOS, tenía conocimientos básicos de Swift, ya que tuve la suerte de hacer las prácticas externas en una empresa que trabaja con este lenguaje, pero las cosas que hice fueron mucho más básicas. He tenido documentarme sobre muchos temas e incluso he visitado la empresa de prácticas para preguntar dudas. Me he dado cuenta que muchas cosas que en Android son nativas, en IOS tienes que programarlas desde cero o utilizar alguna librería externa. He tenido que adaptar la aplicación varias veces el cambio más importante fue cambiar de Swift 2 a Swift 3.

A la vez que hemos estado desarrollando el proyecto, se ha ido constituyendo como una “startup” e incluso ha sido apoyada por Movistar. Esto ha supuesto una presión añadida al grupo y, en muchos casos, nos hemos centrado en temas que quizás no eran tan relevantes para el TFG. Se podría haber extraído información de contenido variado, películas, series, documentales, etc. que quizás para el TFG hubiera quedado mejor, aunque no estuvieran todas. En cambio, hemos optado por centrarnos en extraer la información de todas las películas. Actualmente en la base de datos tenemos casi 182.000 películas.

Al ser un trabajo en grupo, con la particularidad de que vivimos en sitios alejados, se han utilizado una serie de herramientas para coordinar y gestionar el proyecto. Las principales son:

- **Trello**: es un gestor de tareas que permite el trabajo de forma colaborativa mediante tableros compuestos de columnas que representan distintos estados.
- **Skype y Hangouts:** para reuniones virtuales.
- **Slack:** es una herramienta de comunicación en equipo, que ofrece salas de chat organizadas por temas, así como grupos privados y mensajes directos. Además, puedes conectarlas con todas las herramientas, “GitHub”, “Trello”, “Skype”, “Google drive”, etc.

En general creo que ha sido una experiencia bastante positiva, siempre un trabajo en grupo es mucho más difícil de gestionar y coordinar, pero creo que se aprende mucho más. Al fin y al cabo, la realidad es que, en una empresa vamos a tener que trabajar en grupo y lo más normal es que trabajes con compañeros que no se encuentre físicamente en el mismo sitio; creo que nuestra experiencia se acerca mucho a la realidad de trabajo.

En conclusión, creo que hemos entregado un proyecto bastante completo, donde se ha desarrollado aplicaciones nativas de las plataformas móviles más importantes, además de la aplicación web. He adquirido una gran cantidad de conocimientos, no solo de por los lenguajes y tecnologías utilizadas, sino por la experiencia del trabajo en grupo. Me han encantado el lenguaje “Swift” y su nuevo paradigma de programación; he disfrutado desarrollando la aplicación de IOS.

6.2. Trabajo futuro

- **Completar datos:** Ahora mismo solo tenemos datos de películas, podríamos añadir información de series, documentales, etc.
- **Búsqueda Mejorada:** El sistema de búsqueda actual es muy básico, la idea sería mejorarlo, añadiendo la posibilidad de filtrar por duración, género, puntuación, actor, director, etc.
- **Listas personalizadas:** Añadir la posibilidad de que el usuario, pudiera crear sus propias listas para organizar las películas a su gusto.
- **Puntuación del usuario:** El sistema muestra puntuaciones externas y una media de ellas, pero no permite al usuario puntuar la película, se podría añadir esa posibilidad.
- **Sistema de recomendaciones:** Sería interesante que la aplicación fuera capaz de recomendar películas según el gusto del usuario, basándose en las clasificaciones que se han realizado en la aplicación. Para ello se tendría que desarrollar un sistema de recomendaciones que tuviera en cuenta todos los parámetros para proporcionar la mejor recomendación posible.

- **Notificaciones personalizadas:** Ahora mismo el sistema no proporciona ningún tipo de notificación. Sería interesante que notificara cuándo se añade una nueva película, si tu actor favorito ha participado en algún estreno, etc.
- **Tráiler y enlaces:** Sería interesante que en la vista de detalle se pudieran ver los “tráilers” de las películas e incluso se proporcionaran enlaces a plataformas como Netflix, Google Play, Amazon, iTunes para ver la película elegida.

Bibliografía

[1] Apple Inc. (2016). *Develop Apple*. Obtenido de Develop Apple: https://developer.apple.com/xcode/

[2] Apple Inc. (2016). *Developer Apple*. Obtenido de Developer Apple: https://developer.apple.com/swift/

[3] Apple Inc. (2016). *Developer Apple*. Obtenido de Developer Apple: https://developer.apple.com/videos/play/wwdc2016/219/

[4] Apple Inc. (2016). *Developer Apple*. Obtenido de Developer Apple: https://developer.apple.com/library/content/documentation/UserExperience/Conceptual/AutolayoutPG/AnatomyofaConstraint.html#//apple_ref/doc/uid/TP40010853-CH9-SW1

[5] Carthage. (2016). *github*. Obtenido de github: https://github.com/Carthage/Carthage

[6] Christie, T. (2011-2016). *Django REST framework*. Obtenido de Django REST framework: http://www.django-rest-framework.org/

[7] django. (2005-2016). *djangoproject*. Obtenido de django The web framework for perfectionists with deadlines: https://www.djangoproject.com/

[8] Driessen, V. (2010). *nvie*. Obtenido de nvie: http://nvie.com/posts/a-successful-git-branching-model/

[9] Git. (2016). *Git*. Obtenido de Git: https://git-scm.com/

[10] GitHub Inc. (2016). *Help Github*. Obtenido de Help GitHub: https://help.github.com/

[11] Marqués, A. (2013). *asiermarques*. Obtenido de asiermarques: http://asiermarques.com/2013/conceptos-sobre-apis-rest/

[12] Percolate, 5. M. (2015). *Percolate.* Obtenido de Percolate: http://read.prclt.com/50-Most-Important-Mobile-Charts.pdf

[13] PostgreSQL. (2009-2013). *PostgreSQL-es*. Obtenido de PostgreSQL-es: http://www.postgresql.org.es/sobre_postgresql

[14] Python, F. S. (2001-2016). *Python*. Obtenido de Python: https://www.python.org/

[15] Richardson, L. (2004-2015). *Crummy* . Obtenido de Crummy: The Site : https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[16] Wikipedia®. (2016). *Wikipedia*. Obtenido de Wikipedia la enciclopedia libre: https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador

[17] ZenithOptimedia. (2015). *ZenithMedia*. Obtenido de Zenith The ROI Agency: http://www.zenithmedia.com/internet-use-drive-1-4-increase-media-consumption-2015/